UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA DE SOFTWARE

Análisis inteligente de los sensores de un automóvil

Intelligent analysis of car's sensors

Realizado por
Francisco Jiménez Aguilera

Tutorizado por
Enrique Alba Torres
Gabriel Jesús Luque Polo

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2020

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

ANÁLISIS INTELIGENTE DE LOS SENSORES DE UN AUTOMÓVIL

INTELLIGENT ANALYSIS OF CAR'S SENSORS

Realizado por
Francisco Jiménez Aguilera

Tutorizado por
Enrique Alba Torres
Gabriel Jesús Luque Polo

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2020

Fecha defensa: octubre de 2020

Resumen

Los vehículos son a día de hoy sistemas cada vez más complejos que incorporan múltiples sistemas electrónicos que controlan la mayor parte del vehículo. Esos sistemas se comunican mediante una red de buses llamada CAN bus (Controller Area Network bus) y que pueden ser accedidos por un puerto de diagnóstico a bordo (OBD, por su siglas en inglés). De hecho, este puerto es obligatorio en Europa desde 2000-2005 (dependiendo del tipo de vehículo) debido a la Directiva 98/69/CE.

El objetivo de este proyecto es crear un sistema compuesto por múltiples servicios que, en primer lugar, permita poder gestionar *datasets* que contengan los datos sobre los sensores recogidos a lo largo de un recorrido para así luego poder gestionarlos: poder añadir nuevos *datasets,* visualizar su información, poder eliminarlos...

En segundo lugar se pretende poder visualizar dichos datos por medio de diagramas donde se muestra la información de cada sensor de forma detallada e interactiva.

En tercer lugar se quiere añadir un módulo de clasificación por medio de algoritmos de *Machine Learning* como *k-means* y SVM, así como guardar la información de estas clasificaciones y poder consultarla en forma de gráficos.

Finalmente, se pretende crear un módulo de predicción que, usando una red neuronal, permita realizar predicciones sobre un sensor concreto y se pueda consultar los datos de dicha predicción.

Palabras clave:
OBD, inteligencia artificial, sensor, coche, predicción

Abstract

Nowadays, vehicles are increasingly complex systems that incorporate several electronic systems that control most of the vehicle. These systems communicate through a bus network called CAN bus (Controller Area Network bus) and can be accessed through an on-board diagnostic (OBD). In fact, this port is mandatory in Europe since 2000-2005 (depending on the type of vehicle) due to Directive 98/69/EC.

The purpose of this project is to build a system composed of multiple services that, first, allows to be able to manage datasets that contain the data about the vehicle's sensors collected along a trip and be able to manage them: add new datasets, visualize their information, remove them...

Second, it is intended to visualize the sensor's data through diagrams where the information of each sensor is shown in a detailed and interactive way.

Third, it is intended to add a classification module using Machine Learning algorithms such as *k-means* and SVM, as well as save the information of the classifications and be able to access it through diagrams.

Finally, it is intended to add a prediction module that performs predictions on a specific sensor from the dataset using neural networks.

Keywords:
OBD, artificial intelligence, sensor, car, prediction

Índice

1

Introducción

En este primer capítulo se planteará el ámbito de este trabajo de fin de grado, los objetivos que se plantean cubrir y una breve descripción de la organización y contenido de este documento.

1.1 Motivación

A día de hoy nos encontramos con vehículos con una gran variedad de sistemas electrónicos incorporados que se encargan de su monitorización y control. Estos vehículos ofrecen un puerto de diagnóstico a bordo (OBD) que nos permite acceder al valor de los sensores situados a lo largo del vehículo.

Este puerto se usa principalmente para controlar los máximos de emisiones de los vehículos, así como para detectar fallos en alguna zona del vehículo en talleres de mecánica. Es un sistema muy útil puesto que en cuestión de segundos podemos obtener información precisa sobre el estado de cada elemento del vehículo, ahorrando muchas horas de trabajo manual e investigación.

No obstante, a toda esta cantidad de datos se le puede sacar aún más partido mediante un análisis inteligente de los mismos. La idea de este proyecto es poder crear un sistema que, aparte de poder acceder y visualizar los datos para cada sensor, se pueda hacer un análisis inteligente de los mismos para poder establecer patrones de conducción o recorrido, así como crear predicciones de sensores relevantes tales como el nivel de combustible o la temperatura.

1.2 Objetivos

Viendo la necesidad de aportar un valor añadido a la recolección de datos de los sensores de un vehículo a lo largo de un recorrido, el objetivo de este proyecto es crear un sistema capaz de poder tanto consultar el valor para cada sensor como poder analizar

dichos datos por medio de algoritmos de inteligencia artificial capaces de encontrar patrones y poder clasificar estos datos y crear predicciones sobre ellos.

En primer lugar, se ofrecerá un panel que permita poder gestionar *datasets* con los datos capturados del recorrido de un vehículo: añadir nuevos *datasets*, visualizar su información, eliminarlos, etc. A su vez, para poder consultar información más detallada de cada *dataset*, se creará una sección para poder visualizar los datos de estos sensores por medio de gráficas.

En segundo lugar, se pretende hacer un análisis inteligente de los datos. El puerto OBD ofrece multitud de información sobre diversos sensores: revoluciones del motor, posición del acelerador, presión del turbo, cantidad de combustible... Se tendrá que hacer un estudio sobre los mismos para tener en cuenta los más relevantes y poder obtener conclusiones más precisas. Por ello se hará uso de una herramienta de análisis de componentes principales ya que al disponer de un número considerable de sensores y estar relacionados entre ellos, se deben escoger los principales y eliminar información redundante.

Una vez simplificados los datos, es hora de aplicar técnicas de *Machine Learning* sobre ellos. Muchos de estos datos no vienen etiquetados, es decir, no se les ha aportado ningún sentido, como por ejemplo podría ser modo de conducción o el tipo de ruta que se efectúa en cada uno. Por ello, se aplicarán algoritmos de aprendizaje no supervisado capaces de encontrar un patrón entre ellos y darle un sentido.

Por otro lado, podemos tener datos ya etiquetados. Esto nos puede permitir crear una herramienta mediante técnicas de aprendizaje supervisado que ofrezcan la posibilidad de clasificar nuevos datos de entrada en los grupos o categorías que ya están definidas.

Finalmente, se creará un módulo que permita crear predicciones sobre los datos ya analizados por medio de redes neuronales.

1.3 Estructura de la memoria

La memoria está compuesta por 9 capítulos, 2 apéndices y las referencias bibliográficas utilizadas durante el trabajo:

1. **Introducción:** informa sobre qué trata el proyecto, su interés, así como la estructura que sigue.

2. **Tecnologías:** explica los lenguajes de programación, *frameworks* y herramientas que se han usado para el desarrollo de cada servicio.

3. **Inteligencia Artificial:** se ofrece una introducción básica sobre los diferentes tipos de algoritmos de inteligencia artificial que existen dentro del *Machine Learning* así como los que se han usado en el proyecto.

4. **Metodología software aplicada:** se expone la metodología utilizada durante el desarrollo del sistema, sus características, las fases que lo componen y la herramienta de planificación utilizada para cada tarea.

5. **Análisis del sistema:** se describen los requisitos funcionales y no funcionales definidos para el sistema, así como los casos de uso que surgen a partir de ellos.

6. **Modelado y diseño:** se explica cómo se han modelado y diseñado las diferentes partes que componen el sistema por medio de diagramas.

7. **Implementación y pruebas:** se hace un recorrido por las herramientas que se han usado para el desarrollo del sistema, así como de las tecnologías usadas, la estructura que se ha utilizado, los problemas encontrados para la implementación de cada servicio y las diferentes pruebas realizadas para comprobar el correcto funcionamiento del sistema.

8. **Resultados y experimentos:** se describen una serie de casos aplicados para ver la utilidad de los modelos de Inteligencia Artificial en referencia a los automóviles.

9. **Conclusiones y líneas futuras:** se describen las conclusiones tras haber finalizado el proyecto y las mejoras que se le podrían añadir para continuar con el desarrollo del sistema.

10. **Bibliografía y referencias:** se describen todas las referencias a libros o páginas webs utilizadas para consultar información sobre las tecnologías usadas.

11. **Apéndices:** se exponen instrucciones de instalación para los servicios que forman el sistema y un manual de usuario.

2

Tecnologías

Tras indicar el ámbito de este trabajo fin de grado en el capítulo anterior, ahora en este se describirán las diferentes tecnologías software utilizadas. En este TFG se han considerado diferentes lenguajes de programación (Go [P12], Python [G16], Typescript [M19]…) atendiendo a las necesidades específicas de los diferentes componentes que forman nuestra aplicación. En las siguientes secciones describiremos brevemente esos lenguajes, qué bibliotecas o *frameworks* específicos se han utilizado y otros elementos auxiliares necesarios como sistemas de gestión de versiones (Git [CS18]), gestores de bases de datos (Postgres [T20]) o entornos de desarrollo (Visual Studio Code [M20]).

2.1 Go

Go (Golang) es un lenguaje de programación impulsado por Google. Publicado como un proyecto de código abierto, poco a poco ha ido adquiriendo una comunidad bastante robusta. Entre sus principales características se encuentra la concurrencia, tipado estructural, el recolector de basura, un robusto sistema de dependencias y ser multiplataforma. En resumen, un lenguaje que nos ofrece simplicidad, confianza y eficiencia a la hora de programar.

El interés de usar Go en este proyecto se ha visto impulsado por su rápido crecimiento y popularidad en el sector, ya que cuenta con el respaldo de multitud de empresas tecnológicas (Google, Netflix, GitHub [W20c]) formando parte de su infraestructura. A esto se suma que es un lenguaje muy limpio y fácil de leer y su enorme eficiencia. A diferencia de otros lenguajes, al ser más moderno su desarrollo ha ido ligado en el aumento de núcleos en las CPUs y ha permitido que sea mucho más escalable que

lenguajes más antiguos, permitiéndonos aprovechar al máximo los recursos hardware de nuestra máquina y consiguiendo resultados impresionantes.

Para el desarrollo nos hemos servido de dos *frameworks*, uno que facilita la creación de *back-ends* para servicios web (Gin [G20]) y otro para el manejo de base de datos (Gorm [J20]). En las siguientes subsecciones se darán algunos detalles de los mismos.

2.1.1 Gin

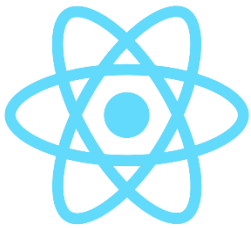
Gin es un *framework* web escrito en Go. Cuenta con multitud de funcionalidades como *routing*, *middleware*, *rendering*... lo cual nos permite crear aplicaciones web y microservicios de forma simple y rápida.

2.1.2 Gorm

Gorm es una librería ORM escrita para Golang que nos permite trabajar con base de datos relacionales de manera sencilla y reduciendo la complejidad de las consultas SQL de nuestra aplicación.

2.2 ReactJS

ReactJS [R20] es una librería de código abierto desarrollada por Facebook que se ha posicionado actualmente como la tecnología más popular para desarrollar aplicaciones web. Se desarrolló bajo la creación de componentes, los cuales se combinan entre sí para formar la aplicación web. Dichos componentes combinan HTML [W20], CSS [W20] y código JavaScript [H11].



En este proyecto ha sido elegido como el principal pilar para nuestro cliente web puesto que permite crear una interfaz gráfica bastante atractiva y rápida al contar con multitud de librerías gráficas con muchos componentes ya realizados. Sigue una estructura muy ordenada y escalable, lo cual nos permitirá trabajar de forma más cómoda. A su vez, se cuenta con el respaldo de una comunidad muy activa, lo cual significa que tendremos muchos recursos que usar a la hora de querer implementar una nueva funcionalidad o resolver algún problema que encontremos.

Además, se ha utilizado Yarn [Y20] para manejar las dependencias de nuestro proyecto; TypeScript, un lenguaje de programación que mejora el manejo de datos y entidades de nuestro sistema; y finalmente Material UI, una librería gráfica para facilitar el desarrollo del cliente. A continuación, serán descritas dichas tecnologías.

2.2.1. Yarn

Yarn es un instalador de paquetes y gestor de dependencias impulsado por Facebook y Google entre otros, creado para JavaScript. Hace competencia a NPM [N20], actualmente el más usado en la comunidad. No obstante, obtenemos mejoras de rendimiento y seguridad frente a él.

2.2.2. TypeScript

TypeScript es un *superset* tipado para JavaScript, es decir, cualquier código JavaScript funciona con TypeScript. Su principal característica es el tipado estático: variables tipadas, interfaces, genéricos, casting... Es muy útil para trabajar en proyectos que implican el flujo de datos entre diversos servicios y se quiere asegurar que los datos de entrada y salida son los correctos, entre otros factores. A parte, aporta mayor facilidad a la hora de desarrollar código ya que aporta autocompletado de código respecto a JavaScript y previene muchos errores para evitar su posterior análisis.

2.2.3. Material UI

Material UI es una librería usada en ReactJS que nos ofrece una serie de componentes que siguen los principios de Material Design [M20] (normativa de diseño enfocada a Android, pero que también se usa en el desarrollo web y otras plataformas). Esto nos permite mayor facilidad a la hora de desarrollar componentes, consiguiendo una interfaz gráfica decente y reduciendo el tiempo de inversión en el diseño de componentes básicos como pueden ser botones, tablas, etc. Dichos componentes permiten diversas configuraciones, adaptándolos a cualquier escenario que se precise.

2.3 Python

Python es actualmente el lenguaje de programación más popular a nivel mundial, ya que es simple, rápido de desarrollar y cuenta con multitud de librerías para cualquier campo. Entre sus características destaca ser multiplataforma, ser interpretado, soportar un intérprete y ser orientado a objetos.

El uso de Python en el proyecto se ha visto impulsado por la cantidad de librerías matemáticas, de inteligencia artificial, etc. con las que cuenta, lo cual nos va a permitir cumplir nuestros objetivos planteados para el análisis inteligente de nuestros datos. A parte, al ser uno de los lenguajes más populares y usados a día de hoy, no tendremos problema en encontrar guías, tutoriales, incidencias resueltas o documentación para cualquier librería que usemos.

Para facilitar la creación de los diferentes servicios de inteligencia artificial, se ha hecho uso de del *framework* Flask [P20a] que permite crear servicios web en Python fácilmente; y de Sklearn [S20], una librería con multitud de recursos de inteligencia artificial.

2.3.1 Flask

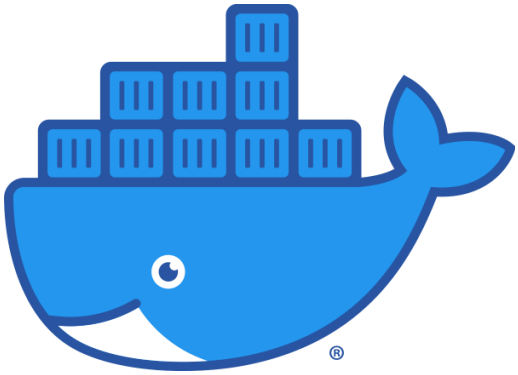
Flask es un WSGI (Web Server Gateway Interface), es decir, una especificación que describe como un servidor web se comunica con aplicaciones web. Destaca por lo simple y rápido que es de montar, lo que permite crear servicios en Python de forma rápida, siempre permitiendo escalar a aplicaciones más complejas.

2.3.2. Sklearn

Sklearn es una librería de *Machine Learning* para Python. Cuenta con multitud de algoritmos entre los que pueden destacar SVM o *k-means*. A parte, se integra muy bien

con otras librerías científicas como NumPy. Con todo ello, permite aplicar técnicas de *Machine Learning* de forma rápida, fácil y robusta.

2.4 Docker



Docker [M16] es un proyecto de código abierto que permite crear contenedores ligeros y portables para que una aplicación pueda ser ejecutada en cualquier plataforma sin importar el sistema operativo que se use o las librerías que haya instaladas. A diferencia de una máquina virtual, Docker no precisa de instalar un sistema operativo para funcionar, sino que funciona sobre el propio sistema donde se ha ejecutado. Esto permite a los desarrolladores poder trabajar desde diferentes plataformas y no preocuparse de instalar ningún software adicional.

Nuestro sistema cuenta con diversos servicios que deben comunicarse entre sí. Por ello, usar Docker en nuestro proyecto va a permitirnos ahorrar mucho tiempo en configurar y levantar cada aplicación, así como comunicarlas entre sí. Para poder trabajar de forma más cómoda, Docker es una alternativa muy útil.

Al trabajar con diferentes servicios, se ha hecho uso de Docker-compose [D20] para su organización y comunicación.

2.4.1 Docker-compose

Docker-compose es una herramienta usada para orquestar diferentes contenedores Docker mediante un archivo de configuración. Esto permite ejecutar diferentes servicios de forma rápida y cómoda, estableciendo las comunicaciones entre ellos. Dicho archivo se puede compartir en un proyecto en grupo, asegurando que todo el mundo utiliza la misma versión de los servicios y sus respectivas dependencias.

2.5 Git

Git es un software destinado al control de versiones, monitorizando los diversos cambios que se realizan sobre una serie de archivos. Aporta gran flexibilidad a la hora de desarrollar en equipo y poder revertir cualquier cambio y volver a la versión anterior del código. A día de hoy, es el más popular en el mundo y es usado por la mayoría de las empresas. Para hacer uso de Git, hemos elegido Github como nuestro servicio web para almacenar nuestros proyectos.

Nuestro trabajo de fin de grado cuenta con diferentes servicios realizados con diferentes tecnologías. Por ello hemos visto necesario usar Github para almacenarlos y llevar un control de los cambios de cada uno. A su vez, permite a los tutores del proyecto ver la

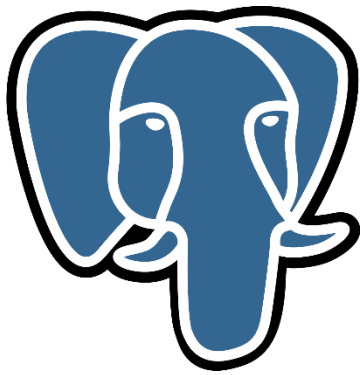
progresión del código y poder abrir incidencias sobre él. Finalmente, y aunque es una función secundaria, nos sirve a modo de copia de seguridad para nuestro sistema, pudiendo acceder de forma remota desde cualquier parte.

2.5.1 Github

Github es un servicio web para almacenar repositorios de software que funcionan bajo Git. De esta manera, podemos compartir nuestro código de forma muy sencilla y poder mejorar el trabajo en grupo, al poder recibir contribuciones de más personas.

A parte de alojar repositorios, cuenta con múltiples herramientas para el trabajo en equipo. Entre ellas podemos destacar una herramienta de revisión de código, un sistema de incidencias, visor de ramas de nuestro proyecto e incluso una funcionalidad que han lanzado recientemente: Github Actions, que permite entre otras cosas la automatización y despliegue de nuestro código (integración continua).

2.6 Postgres



Postgres es un sistema de gestión de base de datos relacional orientado a objetos. Entre sus principales características, destacan que es de código abierto; es multiplataforma, lo que permite trabajar en diferentes entornos; es muy fácil de usar ya que cuenta diversas con herramientas gráficas para ello; y permite manejar un volumen de datos considerable, lo cual destaca entre otras bases de datos relacionales. Para facilitar la administración de los datos, se ha hecho uso de una herramienta gráfica (pgAdmin).

2.6.1 pgAdmin

pgAdmin es una herramienta de código abierta creada para administrar bases de datos Postgres, situándose entre la más popular entre la comunidad. Nos permite visualizar los datos para cada tabla, poder realizar consultas sobre los mismos, etc.

2.7 Visual Studio Code

Visual Studio Code es un editor de texto que poco a poco se ha vuelto muy popular entre los desarrolladores. Cuenta con soporte para la mayoría de los lenguajes de programación actuales. Además, ofrece un sistema de extensiones que lo convierten en un editor robusto y muy útil. Entre sus principales características pueden destacar sus sistemas de autocompletado y *highlighting*, su sistema de depuración y la integración de Git.

El motivo de usar este editor como herramienta para trabajar el código es porque tiene soporte para todos los lenguajes de programación que se han usado y su amplia gama de extensiones que facilitan el desarrollo software.

2.8 Postman

Postman [P20b] es una herramienta que nos permite realizar peticiones a APIs de forma sencilla. Es especialmente útil cuando queremos testear una API REST, puesto que nos permite crear peticiones usando cualquier método HTTP (GET, POST, PUT) y poder configurarlo de diversas formas.

Esta herramienta ha sido especialmente útil en el proyecto a la hora de desarrollar el servidor web, puesto que no hemos necesitado tener las funcionalidades correspondientes en nuestro cliente web para poder probar las diferentes rutas de la API. A parte, para poder probar el servicio de Inteligencia Artificial de manera más cómoda y rápida también ha sido de gran ayuda.

3

Inteligencia Artificial

La IA (Inteligencia Artificial) es la combinación de diferentes algoritmos con el objetivo de imitar las funciones cognitivas de los seres humanos, es decir, poder interpretar datos externos, aprender de ellos y poder generar soluciones a partir de los mismos.

3.1 *Machine Learning* (ML)

Para el desarrollo del TFG se ha optado por seguir una disciplina de la IA denominada *Machine Learning*, que es aquella que se encarga de identificar patrones complejos en multitud de datos. A partir de un análisis de estos, es capaz de predecir futuros comportamientos. A parte, no precisa de la intervención continuada de una persona puesto que van aprendiendo y mejorando a lo largo del tiempo de forma autónoma.

Para adquirir los conceptos básicos y estudiar qué algoritmos, de la amplia variedad que hay, se pueden adaptar al problema que estamos tratando de resolver, se ha hecho uso de un libro [T17] de introducción al *Machine Learning*. Se han establecido las tres principales categorías dentro de ellos, y su diversa forma de tratar los datos de entrada y salida.

3.1.1 Aprendizaje supervisado

El aprendizaje supervisado es aquel que se basa en el uso de datos etiquetados a partir de los cuales se pretenden encontrar patrones para posteriormente poder clasificar datos. Se podrían entender dichas etiquetas como las respuestas a la clasificación del resto de datos del *dataset*.

Dentro de este modelo se encuentran dos categorías, por un lado, la regresión y por otro la clasificación:

- **Regresión:** consiste en encontrar relaciones entre las variables independientes (x) y la variable dependiente (y). De esta manera, se puede obtener un algoritmo para poder predecir alguna cantidad continua. En la Figura 3.1 se puede observar cómo se construye una línea recta que mejor representa a la nube de datos utilizando métodos matemáticos para minimizar la distancia de cada punto a la resta. De esta manera, a partir de x podemos obtener y.

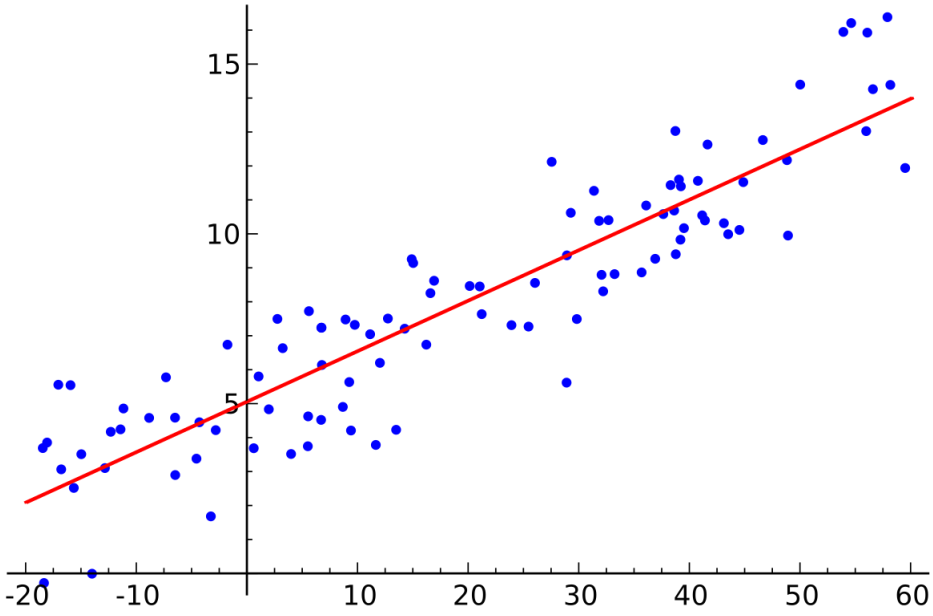


Figura 3.1 Nube de puntos con su respectiva línea de regresión.

- **Clasificación:** este algoritmo encuentra diversos patrones entre los datos para poder clasificar los elementos en diferentes grupos. De esta manera, cuando se quiera clasificar un nuevo dato de entrada se indicará a que grupo pertenece.

Dentro de estas categorías destacan importantes algoritmos como regresión linear, regresión logarítmica, redes neuronales o Máquinas de Vector Soporte. En el TFG se ha elegido tanto las Máquinas de Vector Soporte como las redes neuronales.

Las Máquinas de Vector Soporte o SVM (*Vector Support Machines*) es un algoritmo de aprendizaje supervisado enfocado a la clasificación en este caso. El nombre de vector soporte lo recibe por el margen máximo de separación del hiperplano que separa las diferentes categorías o clases. Se puede observar un pequeño ejemplo en la Figura 3.2 de dicha clasificación.

En la aplicación del TFG se ha utilizado para a partir de un conjunto de datos de sensores ya etiquetados, se puedan clasificar en diferentes clases para posteriormente meter nuevos datos de entrada y poder predecir a qué categoría pertenecen. Si tenemos datos clasificados por tipos de trayectos, podemos determinar a partir de nuevos datos de entrada si se corresponde con un trayecto entre ciudad o una larga distancia.

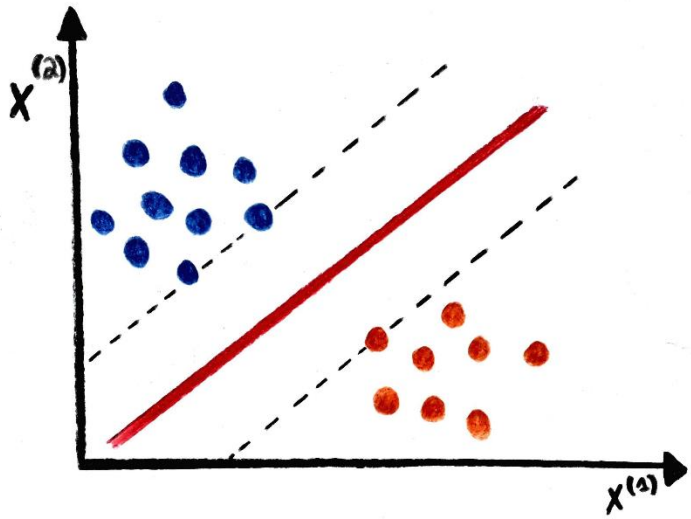


Figura 3.2 Diagrama de una clasificación mediante SVM.

Las redes neuronales se corresponden con un conjunto de nodos, denominados neuronas artificiales que están conectados entre sí y se envían señales. Dichos nodos están jerarquizados en diferentes capas como se puede observar en la Figura 3.3. Cada uno de los nodos posee un peso, un valor que interacciona con los datos de entrada en cada fase. El conjunto de todas estas operaciones desemboca en una salida, un valor que ha sido modificado a través de los datos de entrada y pesos como previamente se ha dicho.

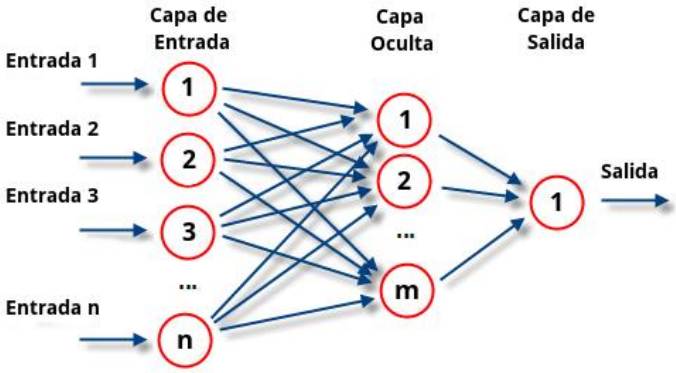


Figura 3.3 Esquema de una red neuronal genérica.

Para entrenar la red neuronal y que cada vez se obtengan datos de salida más cercanos a los resultados que pretendemos obtener, se van modificando los pesos hasta obtener el resultado esperado.

En la implementación de las redes neuronales en el TFG, se ha optado por utilizar redes LSTM [L20], que son una categoría dentro de las redes neuronales recurrentes. Las redes neuronales recurrentes implementan bucles en la red de nodos que permiten recordar estados previos y poder usarlos para realizar predicciones. Se podría decir que mantienen una memoria o copia de los datos a lo largo del flujo. Estas diferencias acerca de los diversos tipos de redes neuronales y su flujo se pueden observar en la Figura 3.4.

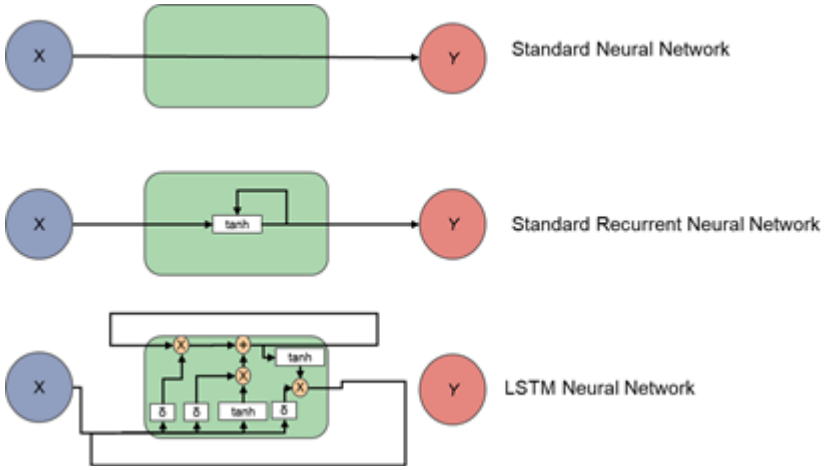


Figura 3.4 Comparación de redes neuronales, redes neuronales recurrentes y redes neuronales LSTM.

3.1.2 Aprendizaje no supervisado

A diferencia del aprendizaje supervisado, el no supervisado consiste en utilizar datos de entrada sin clasificar, es decir, sin etiquetar, para encontrar qué relaciones existen entre dichas características y poder darles sentido a los datos etiquetando cada uno de ellos.

Dentro de este método de supervisión podemos diferenciar dos categorías a su vez: lo que conocemos como *clustering* y la reducción dimensional.

- ***Clustering*:** busca la clasificación por grupos dentro de todo el conjunto de datos. De esta forma compone diversos grupos que comparten características similares. Dentro de esta categoría hemos aplicado el algoritmo *k-means.*

  k-means es un método de agrupamiento que se encarga de dividir todos los datos en *k* grupos en el que cada entrada pertenece al grupo cuyo valor medio es más cercano. Para ello se aplica el método de minimización de la suma de distancias (normalmente cuadráticas) entre cada elemento y el centroide de su grupo o clúster. De esta forma obtenemos conjunto de datos agrupados en diferentes clústeres o grupos como podemos observar en la Figura 3.5, a los que hemos podido darle un sentido.

  En la aplicación del TFG se ha utilizado para dado un *dataset* cualquiera sin estar etiquetado, poder crear categorías para el conjunto de datos provenientes de sensores y poder encontrar un sentido a los mismos.

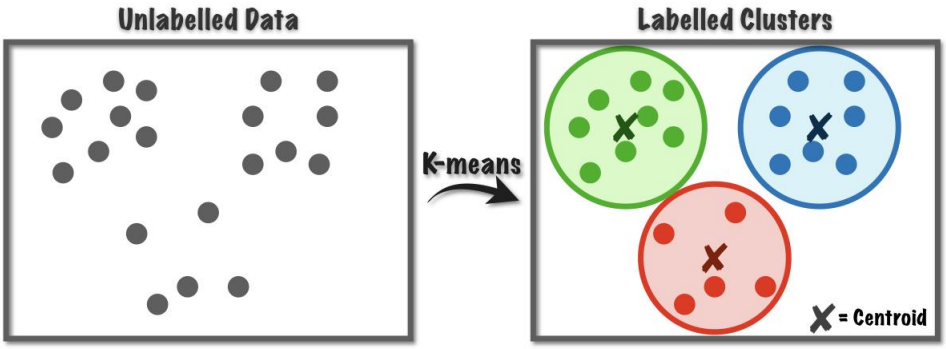


Figura 3.5 Agrupamiento de datos en clústeres.

- **Reducción dimensional:** este método es usado cuando nos encontramos frente a una gran cantidad de datos que se pretenden agrupar, pudiendo procesarlos eliminando la redundancia entre los datos. Este proceso se suele utilizar previamente a la aplicación de otro método de *Machine Learning* para que aumente su rendimiento. Dentro de esta categoría hemos aplicado PCA [P20d].

  PCA (Principal Component Analysis) o Análisis de Componentes Principales es un método que se utiliza para reducir el número de características que usamos para aplicar un algoritmo de *Machine Learning*. Algunas de estas características pueden ser menos influyentes o prescindibles a la hora de la predicción que pueden derivar en *overfitting* en nuestra red neuronal. Para determinar cuáles son más importantes buscamos el grado de participación de estas en el resultado final. En vez de eliminar las características que menos nos interesan, reducimos el espacio de características (o dimensiones) a un menor número. Cada una de las nuevas características es una combinación lineal de todas las características iniciales, por lo tanto, no estamos obviando su participación en el resultado final y conservamos su información útil.

  En nuestro caso de TFG, pretendemos reducir el *dataset* inicial a menos características o dimensiones, para posteriormente poder aplicar el resto de algoritmos como SVM, *k-means* o la red neuronal de predicción. De esta manera aumentamos considerablemente el rendimiento de nuestro sistema.

3.1.3 Aprendizaje reforzado

El aprendizaje por refuerzo (*Reinforcement Learning*) es la última categoría que encontramos dentro del *Machine Learning.* El sistema desarrollado (agente) interacciona con diversas variables y va recibiendo recompensas cuando determina las acciones necesarias para llegar a dicha recompensa. Para ello, busca maximizar dicha recompensa modificando su comportamiento hasta poder ser utilizado para decidir predicciones futuras. El flujo de estas interacciones se puede observar en la Figura 3.6.

Un ejemplo muy común podría ser el juego del ajedrez, en el cual cada vez que el agente gana una partida recibe una recompensa, por lo tanto, trata de encontrar a medida que pasa el tiempo la mayor recompensa posible. De esta manera actualmente existen

modelos de inteligencia artificial capaces de ganar al mejor jugador del mundo de ajedrez en cuestión de horas de aprendizaje.

En el contexto del trabajo de fin de grado no se ha considerado necesario el uso de esta categoría.

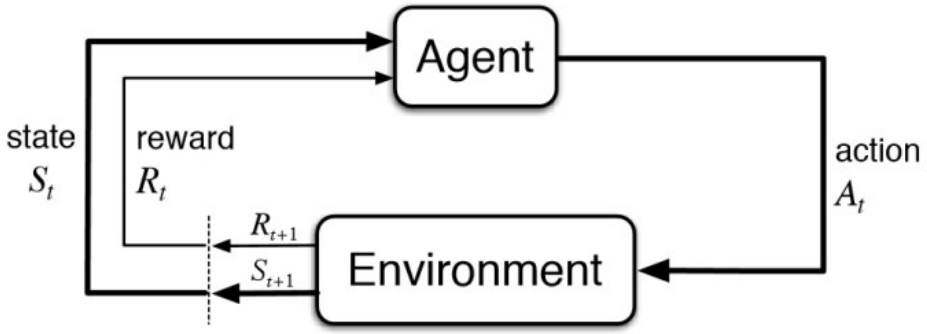


Figura 3.6 Diagrama de acciones de un sistema de aprendizaje reforzado.

4

Metodología software aplicada

Para establecer la metodología software aplicada se han estudiado diferentes alternativas usadas en el ámbito profesional principalmente. Por ello, se ha seguido un desarrollo iterativo e incremental basado en Scrum [S15], pero adaptado a un equipo de reducido de tamaño y al ámbito académico.

En las siguientes secciones se describirá en qué consiste Scrum, cómo lo hemos aplicado en nuestro proyecto, las diferentes fases que lo componen, así como los roles usados en el proyecto. A su vez, también se explicará qué herramienta se ha utilizado para la planificación de las tareas a realizar.

4.1 Scrum

Scrum se trata de una metodología ágil, es decir, se busca una entrega de productos en un periodo corto de tiempo. A dichos periodos se les denomina iteraciones o *sprints*, en los cuales se desarrolla un producto funcional atendiendo a los principales requisitos del cliente. En este caso, dichas iteraciones se han establecido de dos semanas.

4.2 Fases

Cada iteración se divide en las fases que se describen a continuación y pueden verse reflejadas gráficamente en la Figura 4.1.

4.2.1 Planificación de la iteración

La primera fase consiste en la planificación de la iteración. En ella se identifican los requisitos que se desean añadir al producto.

Para poder elegir qué tareas se han de realizar, se hace uso del *Product Backlog,* que consiste en un conjunto de requisitos desarrollados con un lenguaje no técnico y que se jerarquizan atendiendo a su valor en el negocio. Finalmente, el resultado de dicha elección finaliza en la elaboración del *Sprint Backlog*, que se refiere al conjunto de tareas seleccionadas para alcanzar el objetivo del *sprint*.

4.2.2 Reunión diaria

En esta segunda fase, se realizan diariamente reuniones breves (sobre 15 minutos). Para ello cada uno debe explicar al *Scrum Master* y al *Product Owner* lo que ha hecho el día anterior, lo que pretende hacer el mismo día de la reunión y cualquier problema que haya podido surgir.

De esta manera, el equipo está al tanto del progreso de la iteración, qué se ha hecho y qué queda por hacer. A su vez, también permite que los diferentes integrantes del equipo estén al tanto del trabajo que realizan sus compañeros.

Esta fase no ha podido ser aplicada al trabajo de fin de grado debido a que nos situamos en un ámbito académico y no se disponen de los recursos de tiempo necesarios.

4.2.3 Desarrollo

Esta fase, aunque no aparece en el diagrama de la Figura 4.1, se refiere a la implementación de los requisitos acordados en la planificación. Para ello, se debe controlar que solo se realizan los requisitos acordados en ésta y que se cumple con el plazo de tiempo definido para la iteración.

4.2.4 Revisión

En esta fase se revisa todo el trabajo realizado en el desarrollo. En ella participan el equipo de desarrollo, el *Scrum Master* y el *Product Owner*. Se evalúa si los requisitos han sido implementados con éxito y cualquier problema que haya podido surgir, así como cualquier modificación que se pueda realizar.

4.2.5 Retroalimentación

En esta fase también participan personas externas al desarrollo del proyecto. Permiten obtener un *feedback* por parte de los clientes potenciales, que son los que finalmente van a utilizar el producto. Para ello, se suele realizar una demonstración del producto. Tras ello, se puede evaluar lo que se ha hecho bien y las partes que se pueden mejorar.

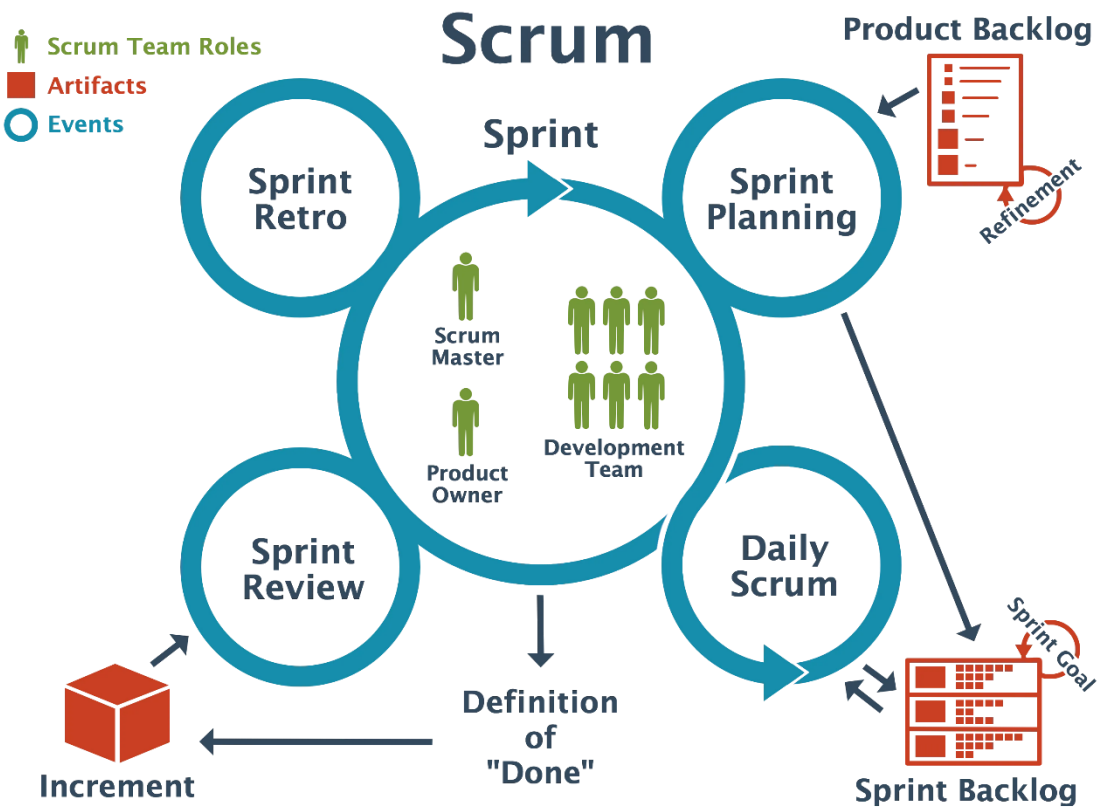


Figura 4.1 Diagrama de las fases de una iteración usando Scrum.

4.3 Roles

Como anteriormente se ha dicho, dicha metodología ha sido adaptada a un equipo de trabajo reducido y al ámbito académico. El rol de *Product Owner* lo han asumido el tutor y cotutor. Los roles de *Scrum Master*, que es el responsable de asegurar que la metodología se está aplicando adecuadamente, y del equipo de desarrollo, se han reducido al alumno.

A continuación, se describen brevemente la función de cada rol.

4.3.1 *Scrum Master*

El *Scrum Master* es la persona que se centra en que el proyecto está avanzando atendiendo a la metodología Scrum. Trabaja conjuntamente con el *Product Owner* para mejorar la metodología de trabajo y resolver cualquier incidencia.

4.3.2 *Product Owner*

El *Product Owner* es la persona que representa a los clientes que han solicitado el producto a desarrollar. Tiene una visión enfocada en el negocio y traduce las necesidades de los clientes al equipo de desarrollo.

4.3.3 Equipo de desarrollo

Conjunto de desarrolladores que de manera conjunta se encargan de la elaboración del producto. Este equipo a su vez se podría organizar en diferentes secciones atendiendo

a las diferentes partes del producto en cuestión, pero esta información queda fuera del dominio de este capítulo.

4.4 Planificación

Finalmente, para la planificación de las tareas seleccionadas en la metodología Scrum se ha utilizado Trello [T20] como herramienta de trabajo. Dicha herramienta es muy útil a la hora de registrar todas las tareas a lo largo del proyecto, ofreciéndonos una interfaz gráfica útil y flexible formada por cajas que almacenan tareas.

Sigue la siguiente estructura: *backlog, to do, doing, done*. No obstante, en el ámbito profesional y cuando se trabaja con más personas se suelen añadir más secciones, como pueden ser *Pull Request* (revisión del código cuando se utiliza Git) o *QA* (si se cuenta con una persona que se dedique a realizar pruebas sobre el producto).

Para organizar mejor cada tarea, se han asignado etiquetas a cada una utilizando diferentes colores, lo cual permite identificarla de manera más fácil. Dichas etiquetas aparecen reflejadas en la Figura 4.2.

Nombre	Descripción
API	Se corresponde con el *back-end* del proyecto.
DB	Se corresponde con todo lo relacionado con el manejo de la base de datos.
Dataset	Se corresponde con todo lo relacionado con hojas de cálculo donde se recogen los valores de cada sensor.
Bug	Se corresponde con cualquier fallo que se haya encontrado en algún servicio y se quiera arreglar.
Chart	Se corresponde con cualquier tarea relaciona con la visualización gráfica de datos.
Client	Se corresponde con tareas para el cliente de nuestro proyecto.
Read	Dicha etiqueta se ha elegido al haber adaptado el proyecto al ámbito académico. Consiste en todos los documentos y libros que se han consultado para la elaboración del TFG.
ML	Se corresponde con todo el trabajo relacionado con la aplicación de ML.

Figura 4.2 Tabla de las diferentes etiquetas del tablón de Trello.

En la Figura 4.3 se puede ver el tablón de Trello en un estado intermedio donde se observan las diferentes tareas etiquetadas y en qué estado se encuentra. Un breve resumen de cada estado es el siguiente:

- ***Backlog:*** consiste en el conjunto de todas las tareas que deben ser añadidas al producto, pero que aún no han sido seleccionadas para ninguna iteración. Las tareas seleccionadas para la primera fase, la planificación de la iteración, provienen de esta sección.

- ***To do:*** en este apartado se encuentran todas las tareas pendientes de realizar en el actual *sprint*.

- ***Doing:*** se corresponde con las tareas que se están realizando actualmente.

- ***Done:*** en este último apartado se recogen todas las tareas que ya han sido realizadas.

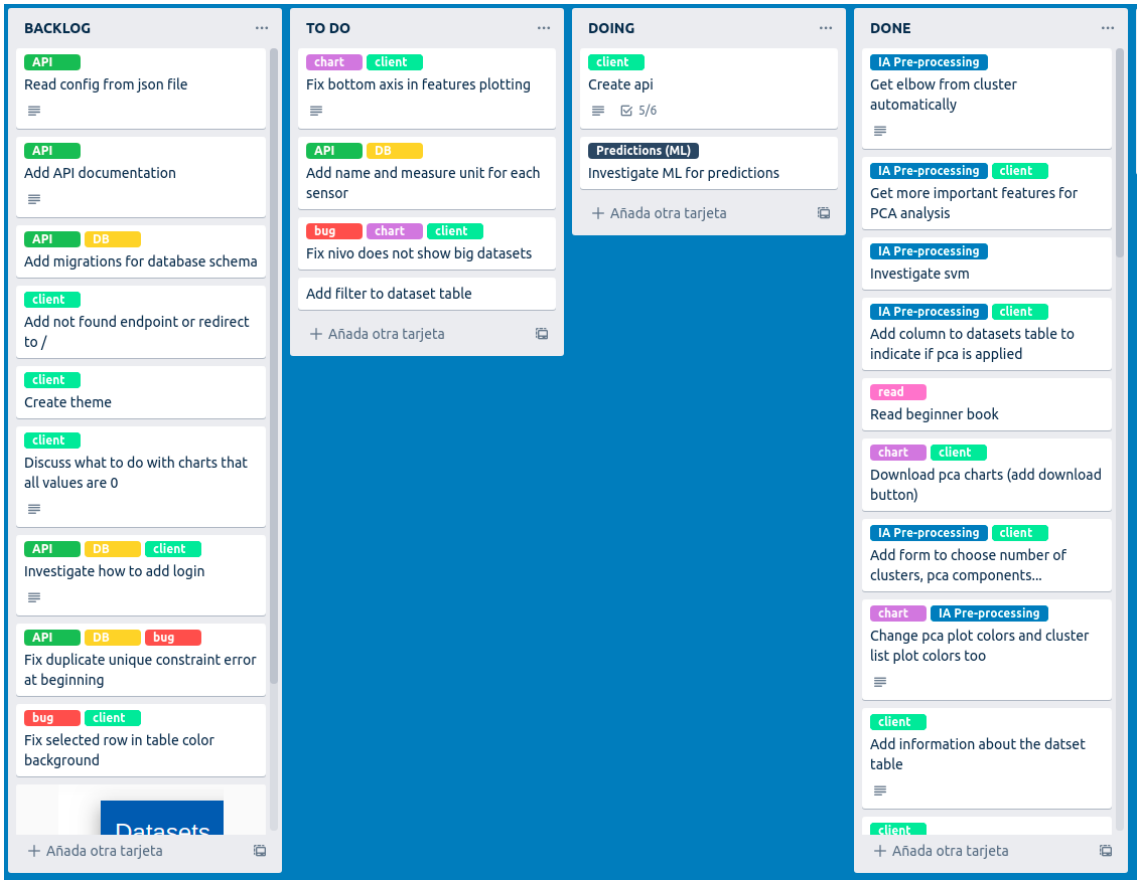


Figura 4.3 Tablón de Trello con los diferentes estados para cada tarea.

5

Análisis del sistema

Antes de empezar la implementación del sistema, es necesario definir qué requisitos debe cumplir éste y todos los casos de uso que se deben implementar. En este capítulo se hará un listado de todos los requisitos definidos (tanto funcionales como no funcionales) así como de los casos de uso acordados acompañados de un diagrama.

5.1 Requisitos funcionales

Los requisitos funcionales son aquellos que describen el comportamiento del sistema, es decir, las acciones que se realizan. A continuación, se detallan los requisitos definidos:

RF1 Gestión de *datasets.*

- **RF1.1** El sistema debe permitir añadir un *dataset.*
- **RF1.2** El sistema debe permitir eliminar un *dataset.*
- **RF1.3** El sistema debe permitir visualizar todos los *datasets.*
- **RF1.4** El sistema debe permitir navegar por la tabla de *datasets.*

RF2 El sistema debe permitir visualizar los datos de los sensores para un *dataset* determinado.

RF3 El sistema debe permitir aplicar *k-means* y SVM sobre un *dataset.*

RF4 El sistema debe permitir realizar una predicción sobre un *dataset.*

RF5 El sistema debe permitir descargar un PDF con todas las gráficas del panel de clasificación.

RF6 El sistema debe permitir descargar un PDF con todas las gráficas del panel de predicción.

RF7 El sistema debe permitir descargar un JSON con los datos para una predicción determinada.

RF8 El sistema debe permitir clasificar un nuevo *dataset* usando el modelo SVM entrenado.

RF9 El sistema debe permitir descargar un *dataset* determinado en formato CSV.

5.2 Requisitos no funcionales

Una vez visto los requisitos funcionales, es momento de hablar de los requisitos no funcionales, aquellos que buscan definir las características y limitaciones que debe seguir nuestro sistema.

RNF1 El cliente web debe ser *responsive.*

RFN2 El sistema debe dar resultados precisos en la clasificación y la predicción.

RFN3 El sistema debe dar resultados en un tiempo razonable.

RFN4 El sistema debe ser sencillo de usar.

5.3 Casos de uso

Una vez vistos los requisitos que se han elaborado para el sistema, es momento de elaborar los casos de uso a partir de ellos. En esta sección se añadirá un diagrama de casos de uso del sistema, así como todos los casos de uso por medio de una tabla explicativa.

Título	**RF1.1** El sistema debe permitir añadir un *dataset.*
Descripción	Se debe permitir que un usuario pueda añadir un nuevo *dataset* por medio de un CSV.
Pre-condición	1. El usuario se encuentra en la página *upload.* 2. El usuario dispone de un CSV válido en su disco duro.
Post-condición	• Éxito: el *dataset* se almacena correctamente. • Error: se muestra una notificación con el mensaje de error respectivo. El *dataset* no se almacena en la base de datos.
Escenario principal	
1. El usuario hace *click* en el área de *drag and drop.* 2. El sistema operativo ofrece una ventana emergente del explorador de archivos.	

3. El usuario selecciona el CSV que quiere añadir. 4. El sistema muestra en la zona de *drag and drop* el CSV que se ha seleccionado. 5. El usuario hace *click* en el botón de enviar (“*send”).* 6. El sistema muestra una barra de cargado mientras se envía la petición a la API. 7. El sistema almacena el *dataset* en la base de datos. 8. El sistema muestra un mensaje de éxito.
Escenarios alternativos
1a. El usuario arrastra el archivo desde el disco duro a la zona de *drag and drop*. 5b. El usuario selecciona otro CSV volviendo al paso 1.

Tabla 5.1 Caso de uso: añadir *dataset*.

Título	**RF1.2** El sistema debe permitir eliminar un *dataset.*
Descripción	Se debe permitir que un usuario pueda eliminar un *dataset*.
Pre-condición	1. El *dataset* se encuentra guardado en la base de datos. 2. El usuario se encuentra en la página principal. 3. El sistema muestra en la tabla el *dataset* que se quiere eliminar
Post-condición	• Éxito: el *dataset* se elimina correctamente. • Error: se muestra una notificación con el mensaje de error respectivo. El *dataset* permanece almacenado en la base de datos.
Escenario principal	
1. El usuario hace *click* en el icono de eliminar. 2. El sistema muestra una ventana emergente para confirmar si de desea eliminar el archivo. 3. El usuario hace *click* en el botón de "confirmar". 4. El sistema elimina el *dataset* de la base de datos. 5. El sistema muestra un mensaje de éxito.	
Escenarios alternativos	
3a. El usuario hace *click* en el botón de "cancelar".	

Tabla 5.2 Caso de uso: eliminar *dataset*.

Título	**RF1.3** El sistema debe permitir visualizar todos los *datasets.*
Descripción	Se debe permitir que un usuario pueda visualizar la principal información de los *datasets*: número de entradas, nombre de las características del *dataset*, fecha de creación y nombre.
Pre-condición	1. Existe al menos un *dataset* en el sistema.
Post-condición	• Éxito: se muestra en la tabla de la página principal los *datasets* existentes en la base de datos. • Error: se muestra un mensaje de error.
Escenario principal	
1. El usuario abre la aplicación web. 2. El sistema muestra en la página principal la información principal de cada *dataset* en una tabla.	

Tabla 5.3 Caso de uso: visualizar *datasets*.

Título	**RF1.4** El sistema debe permitir navegar por la tabla de *datasets.*
Descripción	Se debe permitir que un usuario pueda navegar en la tabla de *datasets* del menú principal a través de un elemento de paginación. Se pueden elegir cuantos elementos mostrar en la tabla.
Pre-condición	1. Existen suficientes *datasets* para poder navegar de una página a otra. 2. El usuario se encuentra en la página principal.
Post-condición	• Éxito: se muestran los *datasets* de una nueva página. • Error: se muestra un mensaje de error.
Escenario principal	
1. El usuario hace *click* en el botón de “siguiente” si no es la última página. 2. El sistema muestra en la tabla los nuevos *datasets* de la nueva página.	
Escenarios alternativos	
1a. El usuario selecciona un nuevo número de *datasets* que mostrar. 2a. El sistema muestra los *datasets* de la página actual cambiando el número al seleccionado en el paso 1a. 1b. El usuario hace *click* en el botón de "anterior" si no es la primera página.	

Tabla 5.4 Caso de uso: navegar por tabla de *datasets*.

Título	**RF2** El sistema debe permitir visualizar los datos de los sensores para un *dataset* determinado.
Descripción	Se debe permitir que un usuario pueda consultar en un diagrama de barras los valores para cada sensor.
Pre-condición	1. El *dataset* se encuentra almacenado en la base de datos. 2. El usuario se encuentra en la página principal. 3. El *dataset* aparece en la tabla principal.
Post-condición	• Éxito: se visualizan gráficamente los valores para cada sensor. • Error: se muestra un mensaje de error.
Escenario principal	
1. El usuario hace *click* en la fila de la tabla correspondiente al *dataset.* 2. El sistema muestra en la parte inferior de la tabla los detalles para el *dataset* seleccionado. 3. El usuario hace *click* en la pestaña de "Dataset charts". 4. El sistema muestra un diagrama de barras con los valores de cada sensor.	

Tabla 5.5 Caso de uso: visualizar gráficamente sensores.

Título	**RF3** El sistema debe permitir aplicar *k-means* y SVM sobre un *dataset*.
Descripción	Se debe permitir que un usuario pueda realizar una clasificación sobre un *dataset* por medio de *k-means* y SVM.
Pre-condición	1. El *dataset* se encuentra almacenado en la base de datos. 2. El usuario se encuentra en el panel de clasificación. 3. El *dataset* aparece en la tabla principal.
Post-condición	• Éxito: se visualizan los resultados de la clasificación. • Error: se muestra un mensaje de error.
Escenario principal	
1. El usuario hace *click* en el botón "Apply classification". 2. El sistema muestra un elemento de cargado. 3. El sistema realiza el análisis.	

4. El sistema muestra un apartado con los resultados obtenidos mediante *k-means* y SVM.

Tabla 5.6 Caso de uso: aplicar clasificación.

Título	**RF4** El sistema debe permitir realizar una predicción sobre un *dataset*.
Descripción	Se debe permitir que un usuario pueda realizar una predicción sobre una característica concreta de un *dataset*.
Pre-condición	1. El *dataset* se encuentra almacenado en la base de datos. 2. El usuario se encuentra en el panel de predicción. 3. El *dataset* aparece en la tabla principal.
Post-condición	• Éxito: se visualizan los resultados de la predicción. • Error: se muestra un mensaje de error.
Escenario principal	
1. El usuario configura el formulario de predicción con los valores deseados. 2. El usuario hace *click* en el botón "Apply prediction". 3. El sistema muestra un elemento de cargado. 4. El sistema realiza la predicción. 5. El sistema muestra los resultados de la predicción.	

Tabla 5.7 Caso de uso: realizar predicción.

Título	**RF5** El sistema debe permitir descargar un PDF con todas las gráficas del panel de clasificación
Descripción	Se debe permitir que un usuario pueda descargar un PDF con todas las gráficas generadas tras el análisis inteligente: diagrama de PCA, *clustering,* SVM...
Pre-condición	1. Lo detalles del *dataset* están cargados en la página. 2. El usuario se encuentra en la pestaña de clasificación.
Post-condición	• Éxito: se descarga un PDF con las gráficas para cada análisis. • Error: se muestra un mensaje de error.

Escenario principal
1. El usuario hace *click* en el botón de "Generate charts document". 2. El sistema comienza una descarga en el navegador web. 3. El sistema guarda el PDF en el disco duro del usuario. 4. El sistema muestra un mensaje de éxito.

Tabla 5.8 Caso de uso: descargar PDF con información sobre una clasificación.

Título	**RF6** El sistema debe permitir descargar un PDF con todas las gráficas del panel de predicción.
Descripción	Se debe permitir que un usuario pueda descargar un PDF con los gráficos generados sobre la predicción, así como la información del experimento.
Pre-condición	1. Lo detalles del *dataset* están cargados en la página. 2. El usuario se encuentra en la pestaña de predicción.
Post-condición	• Éxito: se descarga un PDF con las gráficas para la predicción. • Error: se muestra un mensaje de error.
Escenario principal	
1. El usuario hace *click* en el botón de "Generate document". 2. El sistema comienza una descarga en el navegador web. 3. El sistema guarda el PDF en el disco duro del usuario. 4. El sistema muestra un mensaje de éxito.	

Tabla 5.9 Caso de uso: descargar PDF con información sobre una predicción.

Título	**RF7** El sistema debe permitir descargar un JSON con los datos para una predicción determinada.
Descripción	Se debe permitir que un usuario pueda descargar un archivo JSON con la información sobre una predicción: configuración de la *request*, configuración de la red LSTM y los resultados de la predicción.
Pre-condición	1. Lo detalles del *dataset* están cargados en la página. 2. El usuario se encuentra en la pestaña de predicción. 3. Se ha realizado una predicción sobre el *dataset*.

Post-condición	• Éxito: se descarga un PDF con las gráficas para la predicción. • Error: se muestra un mensaje de error.
Escenario principal	
1. El usuario hace *click* en el botón de "Generate JSON". 2. El sistema comienza una descarga en el navegador web. 3. El sistema guarda el JSON en el disco duro del usuario. 4. El sistema muestra un mensaje de éxito.	

Tabla 5.10 Caso de uso: descargar JSON con información sobre una predicción.

Título	**RF8** El sistema debe permitir clasificar un nuevo *dataset* usando el modelo SVM entrenado.
Descripción	El sistema debe permitir poder clasificar un nuevo *dataset* utilizando el modelo entrenado en el RF3.
Pre-condición	1. Lo detalles del *dataset* están cargados en la página. 2. El usuario se encuentra en la pestaña de clasificación, en el apartado de SVM. 3. Se ha realizado una clasificación sobre el *dataset*.
Post-condición	• Éxito: se descarga el CSV que se ha enviado con una nueva columna con la clasificación realizada por el modelo entrenado de SVM. • Error: se muestra un mensaje de error.
Escenario principal	
1. El usuario hace *click* en el área de *drag and drop.* 2. El sistema operativo ofrece una ventana emergente del explorador de archivos. 3. El usuario selecciona el CSV que quiere añadir. 4. El sistema muestra en la zona de *drag and drop* el CSV que se ha seleccionado. 5. El usuario hace *click* en el botón de "Classify". 6. El sistema muestra una barra de cargado mientras se envía la petición a la API. 7. El sistema realiza la clasificación. 8. El sistema comienza una descarga en el navegador web. 9. El sistema guarda el CSV en el disco duro del usuario. 10. El sistema muestra un mensaje de éxito.	

Tabla 5.11 Caso de uso: clasificar nuevo *dataset*.

Título	**RF9** El sistema debe permitir descargar un *dataset* determinado en formato CSV.
Descripción	Se debe permitir que un usuario pueda descargar los datos del *dataset* almacenado en formato CSV.
Pre-condición	1. El *dataset* se encuentra almacenado en la base de datos. 2. El usuario se encuentra en la página principal. 3. El *dataset* aparece en la tabla de *datasets.*
Post-condición	• Éxito: se descarga el *dataset* en formato CSV. • Error: se muestra un mensaje de error.
Escenario principal	
1. El usuario hace *click* en el icono de "Download CSV" en la fila del *dataset* (tabla). 2. El sistema comienza una descarga en el navegador web. 3. El sistema guarda el CSV en el disco duro del usuario. 4. El sistema muestra un mensaje de éxito.	

Tabla 5.12 Caso de uso: descargar *dataset* en formato CSV.

5.4 Diagrama de casos de uso

A continuación, en la Figura 5.1 se representan los casos de uso por medio de un diagrama de casos de uso.

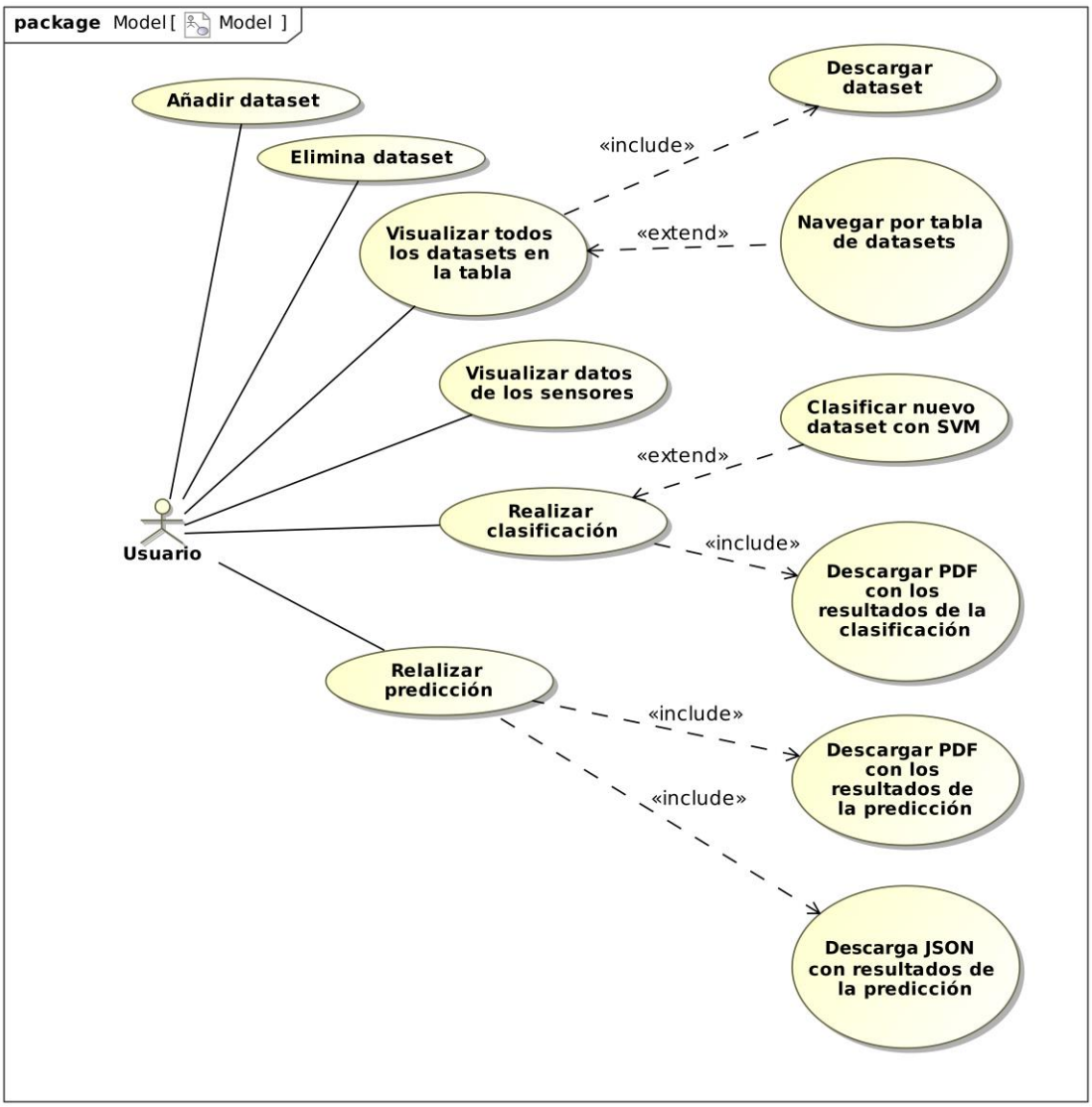


Figura 5.1 Diagrama de casos de uso.

6

Modelado y diseño

Una vez vistos los requisitos del sistema, vamos a mostrar el diseño de nuestro sistema y el modelado de la base de datos que hemos utilizado.

6.1 Modelo base de datos

Se ha diseñado una base de datos relacional. En la Figura 6.1 se muestran las cuatro tablas que representan las distintas entidades del sistema.

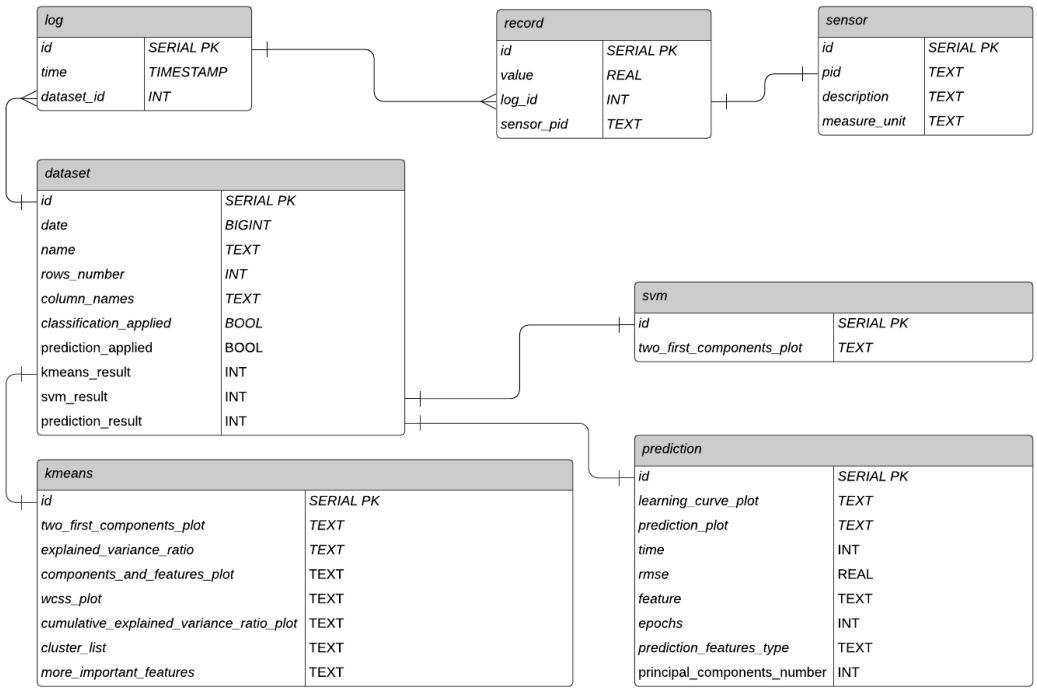


Figura 6.1 Diagrama de la base de datos.

- ***Dataset:*** representan el conjunto de *datasets* que se añaden al sistema. Se guardan datos de carácter descriptivo como el nombre o la fecha de la creación y variables booleanas que controlan si se han realizado o no clasificaciones y predicciones sobre el *dataset*.

- ***Log:*** un *dataset* a su vez está formado por múltiples logs, los cuales guardan la fecha de creación y el *dataset* al que pertenecen. Representan la captura de los datos de sensores para un momento determinado. El *dataset* corresponde a la captura de datos de los sensores cada segundo, por lo tanto, cada entrada de esta tabla se correspondería con una captura específica, que, en su conjunto, forma una serie de datos continuos.

- ***Record:*** un log está formado a su vez por varios *records*, que son los valores registrados para un sensor en un momento concreto. Se guarda tanto el valor del sensor como el log al que pertenecen.

- ***Sensor:*** se corresponde con cada uno de los sensores de nuestro coche. En este caso, tienen una referencia con cada *record*. Se guarda información como el identificador, una descripción de que realiza dicho sensor y la unidad de medida correspondiente.

- ***Kmeans***: es el resultado aplicando *k-means* sobre el *dataset* con el que está relacionado. Almacena gráficas sobre el análisis previo de componentes (ratio de varianza explicada, componentes principales, varianza acumulada...) así como una lista para la clasificación de cada *log* y un listado con las componentes principales.

- **SVM:** es el resultado de aplicar SVM sobre el *dataset* dado. Se almacena un gráfico con los diferentes hiperplanos calculados e información sobre los vectores soporte y la clasificación realizada en su aplicación.

- ***Prediction*:** es el resultado de aplicar la red neuronal LSTM de predicción sobre un *dataset* y un sensor elegido. Se almacenan datos de carácter de configuración general (sensor a predecir, si se van a usar todas, una o PCA para entrenar la red, así como el número de componentes principales elegidos para PCA), la configuración de la red neuronal (épocas elegidas) o los resultados de la predicción (gráfico con la curva de aprendizaje, gráfico de la predicción y el error cuadrático medio y tiempo que ha tardado la predicción).

6.2 Arquitectura del sistema

Se ha desarrollado una arquitectura de microservicios de forma que pueda escalarse a futuras actualizaciones. Las entidades que forman este sistema se representan en la Figura 6.2.

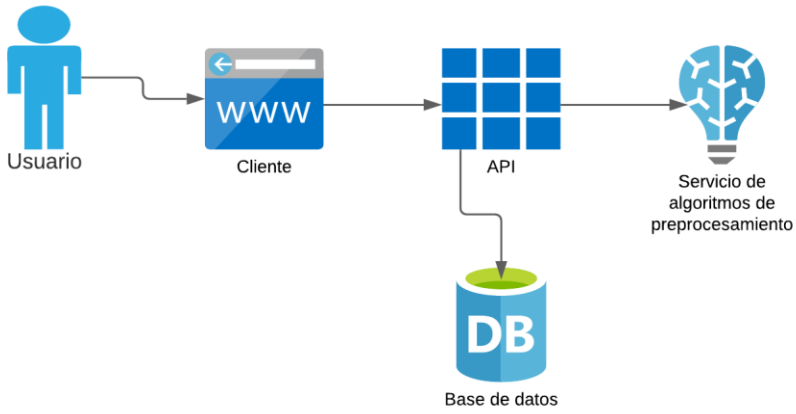


Figura 6.2 Diagrama de la arquitectura del sistema.

- **Cliente:** se corresponde con una aplicación web que permite la interacción con el usuario. En ella se pueden manejar los diversos *datasets* así como consultar todos sus detalles. Por otro lado, contiene funciones para aplicar los servicios de inteligencia artificial.

- **API:** esta entidad se encarga de recibir todas las peticiones del cliente. Puede acceder a los datos guardados en la base de datos y realizar peticiones a los servicios de inteligencia artificial.

- **Base de datos:** en ella se almacena las hojas de cálculo con los datos de los sensores para un recorrido, y posteriormente, los resultados obtenidos tras el análisis inteligente de los datos.

- **Servicio de algoritmos de preprocesamiento:** este servicio se encarga de procesar el *dataset* inicial para eliminar datos redundantes y poder etiquetar los datos (si procede) o crear clasificadores de los mismos.

- **Red neuronal de predicción:** red neuronal que permite crear predicciones a partir de los datos.

7

Implementación y pruebas

Una vez vista la arquitectura de nuestro sistema y las diferentes entidades que lo conforman, es ocasión de hablar de la implementación del proyecto. Para desarrollar los distintos servicios se ha utilizado Visual Studio Code como entorno de trabajo y Github como servicio de control de versiones para cada uno de ellos.

A continuación, se explicarán dichos recursos, la implementación de cada uno de los servicios: tecnologías usadas, la estructura en la que se han organizado y las dificultades encontradas en su implementación, y las pruebas realizadas.

7.1 Entorno de trabajo

Para empezar el proyecto, primero es necesario configurar el entorno de trabajo para trabajar de forma efectiva. Para ello, se han hecho uso de diferentes programas que se describen en las siguientes secciones.

7.1.1 Editor de código

Se ha utilizado Visual Studio Code como editor de código. Al ser un editor muy popular por la gran cantidad de extensiones que contiene, se han añadido algunas útiles como:

- **Prettier:** es un formateador de código compatible con la mayoría de los lenguajes de programación que nos permite formatear el código usando la configuración que deseemos. Es muy útil si se trabaja con un equipo de desarrollo para evitar diferencias en los archivos.

- **Lint:** un *lint* se encarga de detectar errores de código, formateos inválidos, redundancia en el código y muchas más reglas que se quieran añadir. Se ha añadido un *lint* para cada uno de los lenguajes de programación utilizados.

- **Excel Viewer:** una extensión que nos permite visualizar las hojas de cálculo de manera más amigable. Ha resultado muy útil para el manejo de los diferentes *datasets*.

7.1.2 Administrador de base de datos

Para poder administrar la base de datos y poder gestionar los datos de manera más cómoda, se ha hecho uso de pgAdmin, el administrador de bases de datos para Postgres más popular a día de hoy. Nos ha permitido entre otras tareas poder visualizar las tuplas de cada tabla, poder realizar consultas SQL sobre los datos u obtener un diagrama de los mismos.

7.1.3 Testeo de la API

Para poder comprobar que nuestra API funcionaba adecuadamente mientras no se disponía del resto de servicios, como por ejemplo el cliente web, se ha utilizado Postman, una herramienta que nos permite realizar peticiones HTTP sobre nuestra API REST como se puede observar en la Figura 7.1. De esta manera se ha podido testear todas las rutas.

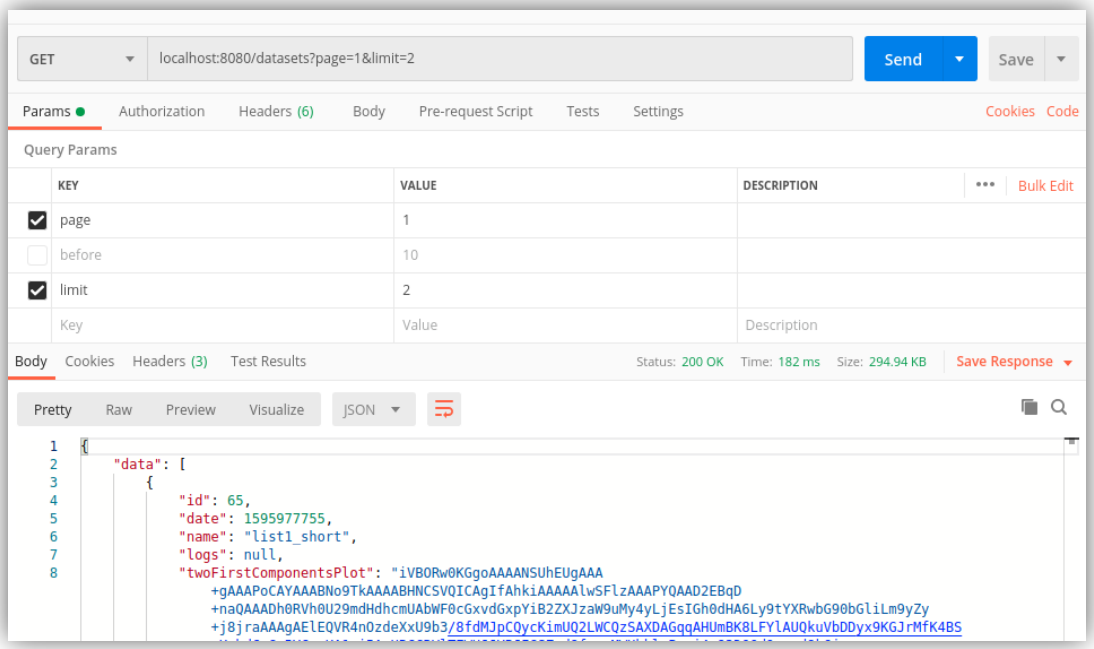


Figura 7.1 Petición GET sobre la API mediante Postman.

7.2 Control de versiones

Como control de versiones para el proyecto se ha utilizado Github. Github es una herramienta de gestión de proyectos y control de versiones de código que se basa en Git.

Para cada uno de los servicios, como se puede observar en la Figura 7.2, se ha creado un repositorio independiente sobre el que se ha construido el código. Para cada tarea se ha creado una rama y posteriormente se ha fusionado sobre la rama principal *master*. Además, cuenta con un documento explicativo denominado README.md en el que se explica brevemente en qué consiste el servicio y un manual de instalación del mismo.

En cada repositorio han sido invitados los tutores, por lo tanto, han podido tener una visión a tiempo real del estado de cada aplicación, pudiendo incluso abrir incidencias o comentar cualquier sugerencia.

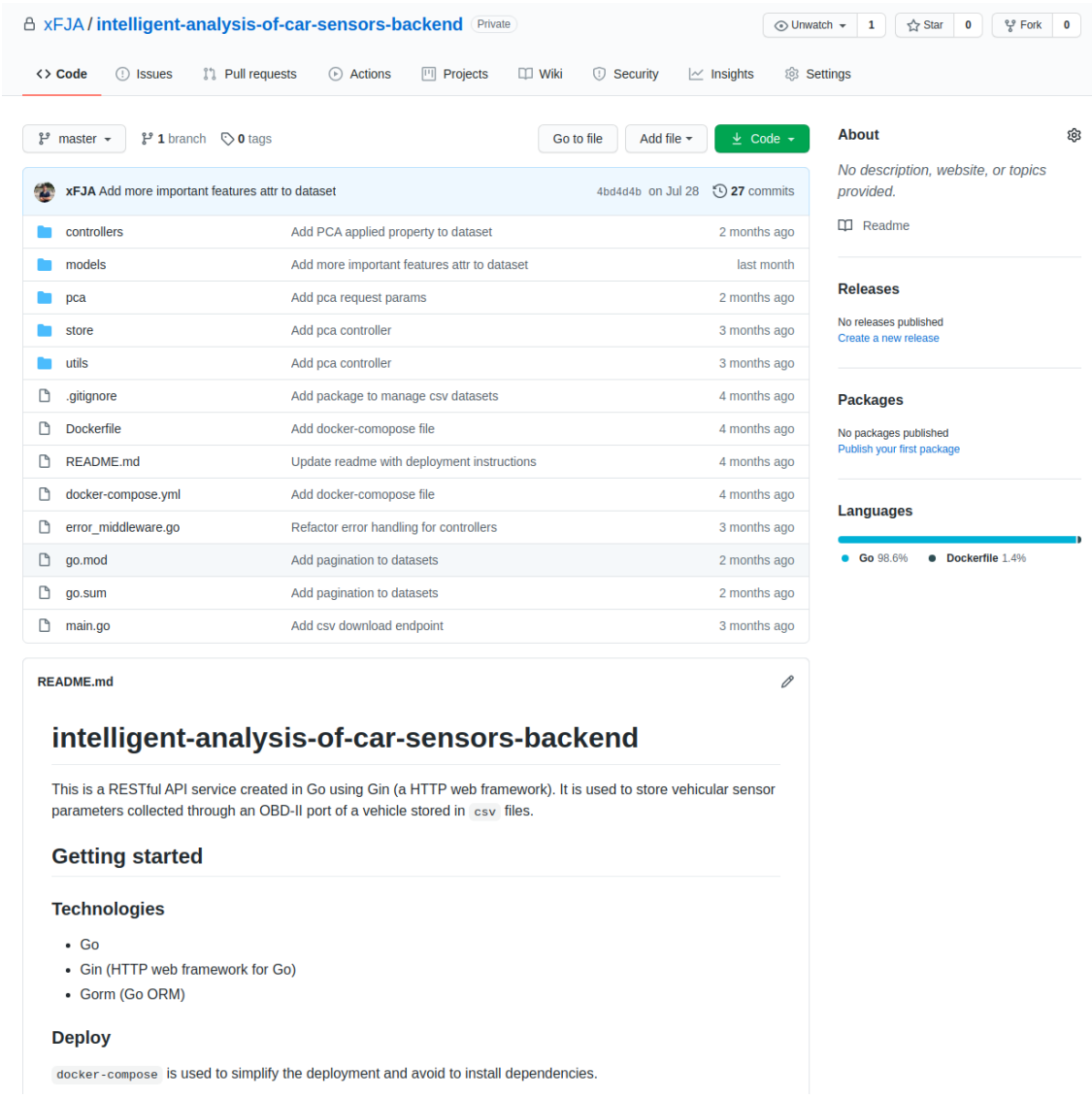


Figura 7.2 Repositorio en Github de la API.

7.3 Fuente de datos

Antes de comenzar la implementación de los diferentes servicios, se ha hecho una búsqueda de una fuente de datos de entrada para el sistema. Para ello, se han buscado *datasets* en las principales plataformas filtrando por proyectos que hayan recogido datos a través del puerto OBD-2 de automóviles.

Finalmente, se ha elegido el siguiente *dataset* https://www.kaggle.com/eron93br/obd2data con el que se han realizado todas las pruebas. No obstante, se ha montado un sistema flexible que permite introducir cualquier tipo de *dataset*. Este *dataset* cuenta con 26 tipos de sensores cuyos valores se han recolectado a través del puerto OBD-2 de un Toyota Etios 2014 1.5, utilizando un *Carloop dev board*, un sistema embebido que permite acceder a los sensores de un coche conectándolo por el puerto OBD-2. Entre los sensores más significativos, se pueden encontrar el sensor de nivel de combustible o la temperatura del líquido refrigerante, los cuales nos permiten hacer predicciones de consumo o temperatura.

Se ha elegido este *dataset* puesto que cuenta con una amplia variedad de diferentes variaciones de *datasets* para elegir: el vehículo realizando un trayecto corto en ciudad, el vehículo realizando distancias largas, el vehículo con el motor encendido, pero sin circular... realizados en diferentes días, lo cual nos permite tener diferentes patrones de conducción que nos va a ayudar a la hora de encontrar clasificaciones más significativas, así como más variaciones en los datos y el recorrido para poder contrastar mejor los datos y tener un contexto más cercano a la realidad.

7.4 Implementación de la API

La primera parte de la implementación de nuestro sistema ha sido la API. En este caso, se ha desarrollado una API REST que es la encargada de recibir las peticiones por parte del cliente e interactuar tanto con la base de datos como los servicios de inteligencia artificial.

A continuación, se hará un repaso de las tecnologías usadas, la estructura en la que se ha organizado, las rutas de la API, cualquier dificultad encontrada y pruebas realizadas.

7.4.1 Tecnologías

Para desarrollar la API se ha utilizado Go. Como previamente se comentó en el capítulo de tecnologías usadas, es un lenguaje de programación que, combinado con Gin, un *framework* web, ha permitido realizar una API REST robusta y eficiente.

Para trabajar de manera más cómoda con la base de datos, se ha utilizado Gorm, que es un ORM (Object Relational Mapping) que permite mapear las entidades de la base de datos y poder realizar sobre ellas consultas sin necesidad de escribir sentencias de SQL.

7.4.2 Estructura

Para establecer la estructura del proyecto, se ha organizado atendiendo a que es una API y se han creado carpetas (a modo de paquetes) para encapsular cada funcionalidad:

- ***Controllers*:** en este directorio se sitúan los métodos encargados de manejar las peticiones HTTP que se reciben. En el ámbito del TFG se han encontrado dos contextos: por un lado, todas las peticiones recibidas por parte del cliente web para la gestión de *datasets*; y, por otro lado, los referentes al análisis inteligente de nuestros datos por medio de los servicios de ML.

- ***Modelos:*** este directorio almacena la definición de las diferentes entidades que conforman nuestro sistema. Para ello se ha hecho uso de *structs*, que es un tipo de datos de Go que permite almacenar valores con un nombre asociado. Para definir un *struct*, como se puede observar en la Figura 7.3, hay que indicar el nombre del campo que deseamos añadir, su tipo y finalmente una etiqueta asociada a él que permite que pueda ser compatible con otros servicios. En este caso, se han añadido etiquetas tanto para configurar Gorm como para asignar el nombre de este campo si la estructura se cómo un objeto JSON.

  El objetivo de definir los modelos, en primer lugar, es para que nuestra herramienta de ORM (Gorm) pueda utilizarlo para crear las tablas de la base de datos. Como previamente se ha indicado, cada campo va asociado con una etiqueta como se observa en la Figura 7.3. En dicha etiqueta se puede indicar si el campo se corresponde con una clave primaria, si es único, etc. No hace falta indicar el tipo de dato en la base de datos puesto que lo infiere del tipado en Go.

  En segundo lugar, estas estructuras nos permiten definir el objeto JSON que se va a enviar a otro servicio. Para ello, como se ha mencionado anteriormente, cada campo del *struct* se etiqueta con la palabra *json* seguida del nombre que tendrá este valor en el objeto JSON. Si este campo se desea omitir en el objeto ya que solo se utiliza en el ámbito de la API, se puede añadir un guion.

```go
// Sensor represents the entity that store the information about
// a car sensor.
type Sensor struct {
    ID          uint   `gorm:"primary_key" json:"id"`
    PID         string `gorm:"unique_index" json:"pid"`
    Description string `json:"description"`
    MeasureUnit string `json:"measureUnit"`
    Record      Record `gorm:"foreignkey:SensorPID" json:"-"`
}
```

Figura 7.3 Modelo de la entidad sensor en Go usando *structs*.

- ***Store:*** en esta carpeta se encuentran todas las funciones asociadas a la gestión del *dataset*, por lo que únicamente contiene un archivo

csv_store.go para ello. En dicho archivo se encuentra un método principal *Load* que dado un CSV y un nombre, es capaz de leer el archivo y desestructurarlo en los modelos previamente definidos. A su vez, también contiene un método *LoadFromFile* que fue usado para hacer pruebas con la API cuando el cliente web no había sido empezado y se precisaba obtener el archivo directamente del disco duro.

- ***Utils:*** en esta carpeta se almacenan todos los métodos que sirven de complemento para nuestro servicio. En este caso, solo se ha creado un archivo *sensors_information.go* que se encarga de cargar desde un archivo *.json* la información sobre cada sensor de nuestro sistema (Figura 7.4) para posteriormente ser usada en nuestro cliente web: descripción del sensor y la unidad de medida usada. Este apartado es necesario ya que estos valores no se pueden obtener de los *datasets* usados.

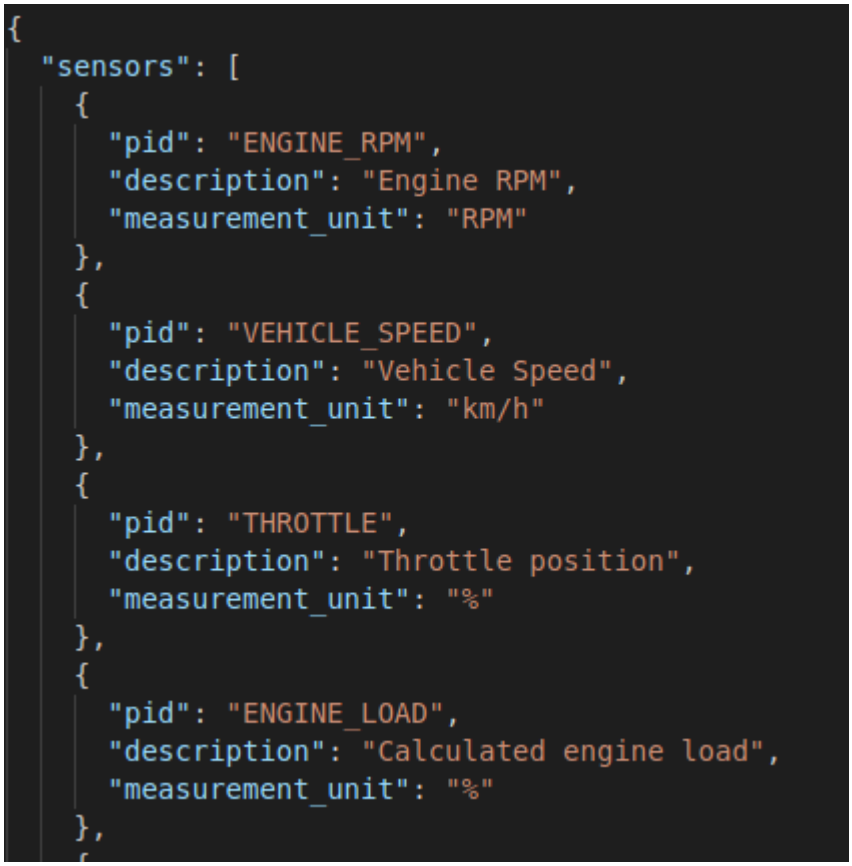


Figura 7.4 Archivo de información sobre cada sensor.

- ***main.go*:** este archivo es el encargado de inicializar la API. En él se crea el *router* que es la entidad de Gin encargada de administrar y manejar todo lo relacionado con las rutas del servicio. Posteriormente se añaden todos los *middlewares* (métodos que se ejecutan en cualquier petición HTTP). En nuestro caso solo se ha contemplado usar *Cors* para permitir recibir peticiones desde el cliente web; uno creado por nosotros, *ErrorHandler* que se encarga se formatear el mensaje de error que se devuelve en un controlador y finalmente un último método que se encarga de mandar la

instancia de la base de datos a todos los controladores, de esta forma aseguramos una única conexión.

A su vez, también se realiza la conexión de la base de datos (previamente a la creación del *middleware* que la utiliza) y se definen tanto los controladores como las rutas de nuestra API.

7.4.3 Rutas

Una vez definida la estructura de nuestro sistema, y especialmente el apartado de controladores, a continuación, se definen las rutas o *endpoints* que son usadas por el *router* y que manejan los controladores. Estas rutas son consumidas por nuestro cliente web.

Ruta	/datasets
Método HTTP	GET
Descripción	Devuelve información acerca de todos los *datasets* del sistema

Tabla 7.1 Definición de ruta GET /datasets.

Ruta	/datasets
Método HTTP	POST
Descripción	Añade un nuevo *dataset* al sistema
***Form* (Formulario POST)**	csv (binary)

Tabla 7.2 Definición de ruta POST /datasets.

Ruta	/datasets/:id
Método HTTP	GET
Descripción	Devuelve la información de un *dataset* así como la de sus entidades relacionadas (los valores para cada sensor y cada *log* (registro)

Tabla 7.3 Definición de ruta GET /datasets/:id.

Ruta	/datasets/:id
Método HTTP	DELETE
Descripción	Elimina un *dataset* del sistema dado su *id*

Tabla 7.4 Definición de ruta DELETE /datasets/:id.

Ruta	/datasets/:id/csv
Método HTTP	GET
Descripción	Devuelve el *dataset* completo en formato CSV

Tabla 7.5 Definición de ruta GET /datasets/:id/csv.

Ruta	/sensors
Método HTTP	GET
Descripción	Devuelve toda la información de cada sensor del sistema (unidad de medida, nombre, código y descripción)

Tabla 7.6 Definición de ruta /sensors.

Ruta	/classify/:id
Método HTTP	GET
Descripción	Aplica una clasificación sobre el *dataset* a través de *k-means* y SVM
Request	• *clusters-number:* número de clústeres que encontrar para *k-means* • *componentes-number:* número de componentes principales para PCA

Tabla 7.7 Definición de ruta GET /classify/:id.

Ruta	/classify-svm
Método HTTP	POST

Descripción	Realiza una clasificación sobre un nuevo *dataset* dado utilizando el modelo SVM entrenado a partir del *dataset* original
***Form* (Formulario POST)**	csv (binary)

Tabla 7.8 Definición de ruta POST /classify-svm.

Ruta	/predict/:id
Método HTTP	GET
Descripción	Realiza una predicción sobre un sensor determinado utilizando el *dataset* original como entrenamiento
Request	• *feature:* característica elegida para predecir • *epochs:* número de épocas elegidas para la red neuronal • *predictions-feature-type: parámetro que determinada si se usan todos los sensores, uno o PCA para el entrenamiento* • *pc-number:* en el caso de elegir PCA, el número de componentes principales

Tabla 7.9 Definición de ruta GET /predict/:id.

7.4.4 Dificultades encontradas

A lo largo del proyecto se han ido encontrando diferentes dificultades que han sido solventadas con éxito:

- **Estructurar aplicación:** al tener que aprender un lenguaje de programación nuevo, se ha hecho un estudio profundo sobre cómo estructurar adecuadamente la aplicación, como documentarla y buenas prácticas de código dentro de la comunidad de Go. Se ha hecho mucho énfasis en generar una estructura atendiendo a buenas prácticas de código.

- **Establecer relaciones en Gorm:** a la hora de indicar qué entidades y cómo están relacionadas, la versión actual de Gorm no soporta aún la inferencia directa de las mismas, especialmente para relaciones de uno a muchos. La 2º versión de este paquete estaba en proceso de desarrollo al inicio del TFG, pero debido al COVID el desarrollador no ha podido terminarla para la fecha prevista. Por ello he tenido que añadir directamente a cada entidad campos para solventarlo, siguiendo siempre buenas prácticas para el modelado de base de datos.

- **Formateo de errores en los controladores:** para formatear el mensaje de error en caso de cualquier fallo en algún controlador, únicamente se enviaba el código de error, pero no el mensaje asociado. Para solventarlo, se ha creado un middleware que en caso de lanzar un error el controlador, obtiene el mensaje asociado a este y lo devuelve en la respuesta HTTP para que el cliente lo reciba adecuadamente y pueda mostrarlo al usuario.

7.4.5 Pruebas

Para desarrollar las pruebas sobre la API, se han definido una serie de *tests* unitarios para comprobar el buen funcionamiento de los bloques de código. Para ello se ha seguido una buena práctica evitando crear funciones extensas y así poder probar cada una de manera independiente.

Para realizar los *tests* unitarios en Go se ha utilizado la librería *testing* que viene incluida de forma predeterminada y *testify/assert* para poder comprobar que una condición determinada es cierta como por ejemplo comparar dos entidades.

Tras consultar la documentación de Go, la convención a la hora de definir *tests* es la siguiente: se crea junto al archivo que se desea testear un nuevo archivo con el mismo nombre y como sufijo *_test.* Por ejemplo, si se pretende testear el archivo *dataset.go* (encargado de gestionar la entidad *dataset*), el archivo de *test* debe llamarse *dataset_test.go*.

Una vez se ha definido el *test* unitario, sobre la raíz del proyecto se ejecuta el comando *go test ./...* el cual ejecutará todos los *tests* existentes en el proyecto, paquete por paquete.

7.5 Implementación del cliente web

Una vez realizada la API, el siguiente paso en la implementación del sistema es desarrollar el cliente web con el que interactuará el usuario.

En las siguientes secciones se hará un resumen de las tecnologías usadas en el cliente y cómo se han combinado, de la estructura que se ha seguido, cómo se ha configurado el diseño, las librerías gráficas referentes a los diagramas realizados, dificultades encontradas en su desarrollo y las pruebas realizadas.

7.5.1 Tecnologías

Para el desarrollo de nuestro cliente web se ha optado por usar ReacJS, una librería JavaScript para construir interfaces de usuario o componentes UI.

Para la gestión de dependencias se ha usado Yarn en vez de NPM, ya que es totalmente compatible con esta y aporta mejoras en rapidez y seguridad.

En cuanto al diseño, se ha elegido Material UI como *framework* para facilitar la creación de componentes y conseguir un mejor resultado.

Finalmente, debido a que se ha desarrollado un sistema con múltiples servicios y entidades, se ha añadido TypeScript sobre la aplicación para tener un mejor manejo de las entidades del sistema, evitando errores y consiguiendo un código más legible y robusto.

7.5.2 Estructura

Dentro de la carpeta *src* dónde se implementa todo el código referente al cliente, se han divido en diferentes directorios atendiendo a la implementación de las peticiones realizadas a la API, los componentes que forman la interfaz gráfica y las interfaces definidas usando TypeScript para cada entidad de nuestro *dataset* y entidades creadas para los componentes. A continuación, se hace un repaso sobre ellos:

- ***api:*** esta carpeta contiene los métodos que realizan peticiones web. En nuestro contexto, solo se ha creado un archivo *api.ts* referente a las consultas realizadas sobre nuestra API.

  Dichos métodos son usados en nuestros componentes para realizar funciones como añadir un nuevo *dataset* al sistema, borrarlo o solicitar un análisis inteligente del mismo. Se ha utilizado *axios* como librería para generar las peticiones HTTP. En la Figura 7.5 se puede observar un ejemplo en el que se realiza una petición para obtener todos los *datasets* del sistema para posteriormente poder visualizarlos. Como parámetros se reciben valores referentes a la paginación que se ha realizado (los datos se muestran en una tabla y se cargan en tandas). Posteriormente se forma la URL con los parámetros de la petición GET y finalmente se maneja la respuesta, que es una promesa.

  Como se puede observar en la Figura 7.5, la promesa está tipada por la interfaz *DatasetsRequest*, por lo tanto, tenemos un control de los datos que nos llegan desde la API.

```
getDatasets = async (
  limit: number,
  page: number
): Promise<DatasetsRequest> => {
  return await axios(`${BASE_URL}datasets?limit=${limit}&page=${page}`).then(
    (response: any) => {
      if (response.status !== 200) throw new Error(JSON.stringify(Response));
      return response.data;
    }
  );
};
```

Figura 7.5 Petición GET sobre la API a través de *axios*.

- ***components:*** está carpeta contiene todos los componentes que forman nuestra aplicación. Un componente es un elemento que genera elementos React (que posteriormente se traduce en HTML y CSS), es totalmente independiente y se puede reutilizar. De esta forma podemos crear por ejemplo un componente que sea un formulario. Este a su vez está compuesto por un componente que es un campo de texto para escribir y otro componente, el botón que se utiliza para enviar el formulario. Finalmente, obtenemos un formulario que a su vez puede ser usado múltiples veces en toda nuestra aplicación.

  Este directorio se estructura en diferentes carpetas (Figura 7.6) para cada componente que se ha creado:

  - ***Bar:*** componentes que generan los diagramas de barras utilizados para visualizar los datos de cada sensor.
  - ***Header:*** componente que forma nuestra cabecera de la aplicación (icono, título, botón para añadir *dataset*...).
  - ***Home:*** componentes que forman la página principal de nuestro cliente.
  - ***Plot:*** componentes que se encargan de generar los diagramas de puntos usados para representar datos del análisis inteligente.
  - ***ClassificationPanel:*** componentes que forman el panel de clasificación.
  - ***PredictionPanel:*** componente que forma el panel de predicción.
  - ***Utils:*** componentes que se han usado por su flexibilidad en varias partes del proyecto.

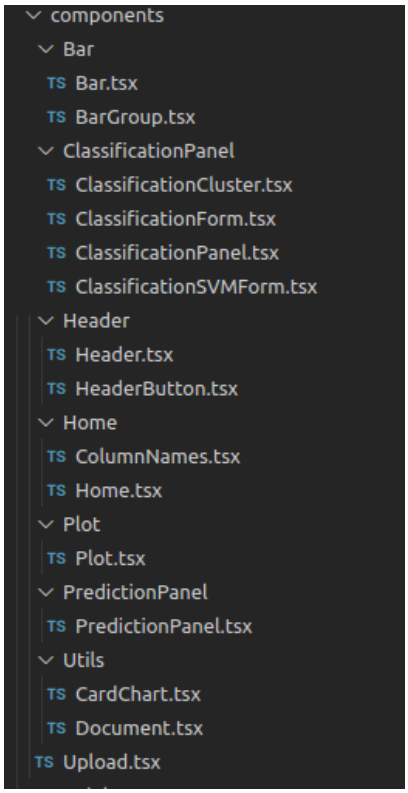


Figura 7.6 Estructura de la carpeta componentes.

Para implementar cada componente se ha decidido utilizar *React Hooks* en vez de clases, puesto que es una nueva funcionalidad que incluyó React en 2017. Ofrece mejoras en rendimiento, aumenta la legibilidad del componente y reduce las líneas de código. Se ha elegido para estar actualizado con la comunidad actual y conseguir un mejor resultado. Los *hooks* se encargan de manejar el comportamiento de nuestro componente. Por ejemplo, un *hook* muy frecuente es el *useState*, que se encargado de crear estados para nuestro componente (se entiende por estado a la memoria que se tiene sobre la instancia del componente que se está mostrando). En este caso, por ejemplo, dentro del ejemplo de crear un formulario, se tendría un estado para almacenar el valor del campo de texto. Este *hook* se encarga de almacenarlo y poder actualizarlo.

A su vez, a los componentes se le puede proveer de propiedades para poder inicializar estados o usarlas como datos para gestionar. Para ello, haciendo uso de TypeScript, se definen interfaces para cada componente donde se establecen los campos que forman estas propiedades y sus tipos. De esta manera tenemos control de que los componentes se están instanciando con las propiedades adecuadas y no hay fallos. Si utilizáramos JavaScript encontraríamos este problema, ya que permite recibir cualquier valor y solo señalaría el error en mitad de la ejecución.

Un ejemplo de *hooks* (*useState* y *useEffect*) y la definición de las propiedades de un componente (*interface Props)* se pueden observar en la Figura 7.7. Se puede ver como para el componente *ColumnNames*, que se encarga de mostrar el nombre de las columnas que forman el *dataset,* recibe como propiedad los nombres en formato *string* y posteriormente, en ambos *hooks* se define el estado que almacena el listado de nombres y su inicialización: dentro de *useEffect e* por medio del método *setNamesList* que inicializa el estado dado un valor.

```
interface Props {
  names: string;
}

export const ColumnNames: React.FC<Props> = (props) => {
  const classes = useStyles();

  const { names } = props;

  const [namesList, setNamesList] = useState<string[]>([]);

  useEffect(() => {
    setNamesList(names.split(","));
  }, [names]);
```

Figura 7.7 Definición de un componente en ReactJS.

- ***models:*** este directorio contiene la definición de todas las entidades que forman el sistema y usadas por los componentes por medio de interfaces de TypeScript. Se ha creado un archivo para cada entidad (*bar.ts, dataset.ts, pdf.ts* y *plots.ts),* y dentro de cada una, se definen las interfaces.

  De esta manera, podemos observar cómo en la Figura 7.8 se define la entidad *DatasetRequest*, que representa el objeto JSON que se recibe cuando se quieren obtener todos los *datasets* del sistema. Ésta a su vez está formada por la interfaz *Dataset*, que contiene la información para cada *dataset* así como un *array* con todos los *logs.* Se puede observar cómo se van anidando todas las estructuras y se tiene un manejo del tipado de todos los datos. De esta forma, aseguramos que todos los datos que nos llegan son correctos, y a su vez, nos permite programar de manera más cómoda ya que el editor de texto no sugiere los campos a la hora de escribir código. Si se usase JavaScript, teniendo un objeto, se podría acceder a cualquier propiedad, lo cual puede derivar a un error. En este caso, TypeScript solo nos sugiere y permite acceder a los campos definidos en la interfaz.

```typescript
export interface DatasetRequest {
  data: Dataset;
}

export interface Dataset extends LightDataset {
  logs: Log[];
  twoFirstComponentsPlot: string;
  componentsAndFeaturesPlot: string;
  explainedVarianceRatio: string;
  wcssPlot: string;
  cumulativeExplainedVarianceRatioPlot: string;
  clusterList: string[];
  moreImportantFeatures: string; // TODO: Use the proper interface here
}

export interface LightDataset {
  id: number;
  date: number;
  name: string;
  rowsNumber: number;
  columnNames: string;
  pcaApplied: boolean;
}

export interface Log {
  id: number;
  time: number;
  records: DatasetRecord[];
}

export interface DatasetRecord {
  id: number;
  value: number;
  sensorPID: string;
}
```

Figura 7.8 Definición de interfaces referentes a la entidad dataset.

7.5.3 Diseño

Para el diseño de los componentes, en vez de utilizar directamente CSS o SCSS como lenguajes para dar estilo web, comúnmente usado en cualquier aplicación web, se ha optado por usar CSS sobre JS utilizando la librería *@material-ui/core/styles* que provee Material UI. De esta forma, tenemos acceso a muchas funcionalidades de React (anidamiento de temas, estilos dinámicos...) a la hora de aplicar estilo a los componentes.

Para ello, como se puede observar en la Figura 7.9 se sigue un patrón para formar una función que nos devuelve el estilo para cada componente. Consiste en un objeto JS que encapsula los diferentes atributos de estilo. A su vez, se hace uso del tema (*theme)* definido para el cliente, lo cual, entre otras cosas, nos permite acceder a la paleta de colores definida o al espaciado, lo cual nos asegura que todos los componentes se forman bajo un tema ya definido y no de forma arbitraria. Por ejemplo, para dar estilo al botón que nos lleva al *Home*, se nombra como *homeButton* y se aplican propiedades que le dan un color al texto, una altura y una anchura.

Posteriormente, en el cuerpo del componente, se instancia este estilo en la variable *classes* y se hace referencia a *homeButton* en la propiedad *className* de cada componente como se puede observar en la Figura 7.10. Esto nos permite acceder fácilmente a los estilos para cada componente.

```
const useStyles = makeStyles((theme: Theme) =>
  createStyles({
    link: {
      textDecoration: "none",
    },
    linkGroup: {
      marginLeft: 50,
      marginRight: theme.spacing(6),
      right: 0,
      position: "absolute",
    },
    button: {
      color: theme.palette.common.white,
      textTransform: "none",
    },
    homeButton: {
      color: theme.palette.common.white,
      height: 64,
      width: 64,
    },
  })
);
```

Figura 7.9 Definición del estilo para un componente usando *styles*.

```
const classes = useStyles();

let history = useHistory();

return (
  <AppBar position="sticky" elevation={12}>
    <Toolbar>
      <IconButton
        edge="start"
        onClick={() => {
          history.push("/");
        }}
      >
        <DriveEtaIcon className={classes.homeButton} />
```

Figura 7.10 Adición de estilo a un componente.

7.5.4 Librerías gráficas

Para poder visualizar los datos de los sensores o del análisis inteligente de los mismos, se ha precisado usar librerías gráficas para elaborar diagramas. Se ha hecho un estudio de las principales usadas actualmente para ReactJS (d3, Paper.js, Nivo, DrawD2...) y finalmente, tras realizar *demos* sobre cada una, se ha elegido Nivo puesto que contiene una amplia gama de diagramas que se adecuan a nuestras necesidades, contiene una documentación muy completa y práctica, y permite mucha personalización.

Nivo es una librería gráfica construida sobre d3 y las librerías de ReactJS que ofrece una amplia variedad de gráficas (nube de puntos, diagrama de barras, grafos, árboles...) para visualizar datos. Ofrece un resultado de diseño muy bueno y genera elementos SVG que nos permiten reescalar las gráficas fácilmente, ideal para una aplicación *responsive.*

En el TFG solo se ha hecho uso de dos componentes: *Bar*, que permite realizar diagramas de barras; y *ScatterPlot*, que nos genera diagrama de nubes de puntos. Ejemplos de ellos se pueden observar en las figuras 7.11 y 7.12.

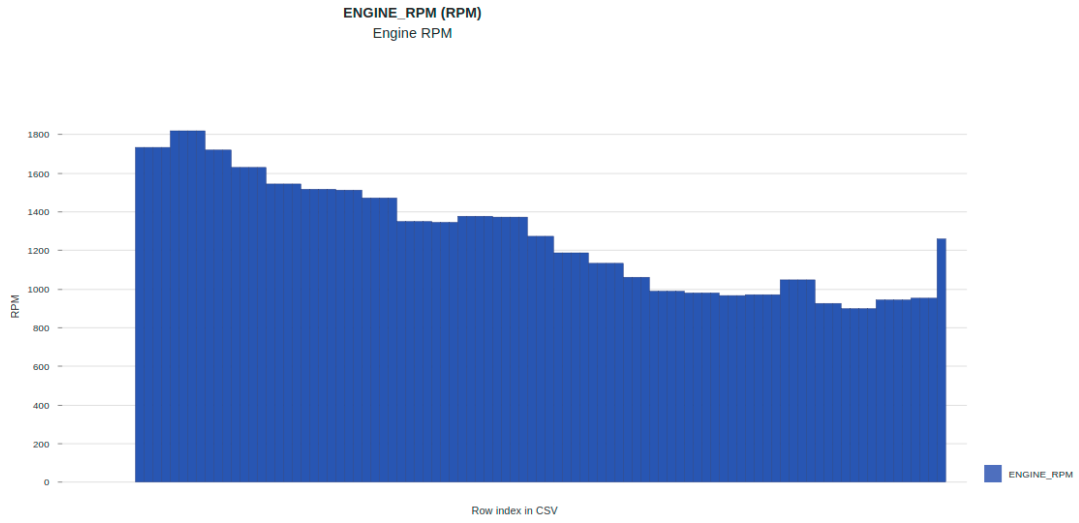


Figura 7.11 Diagrama de barras de los datos de un sensor.

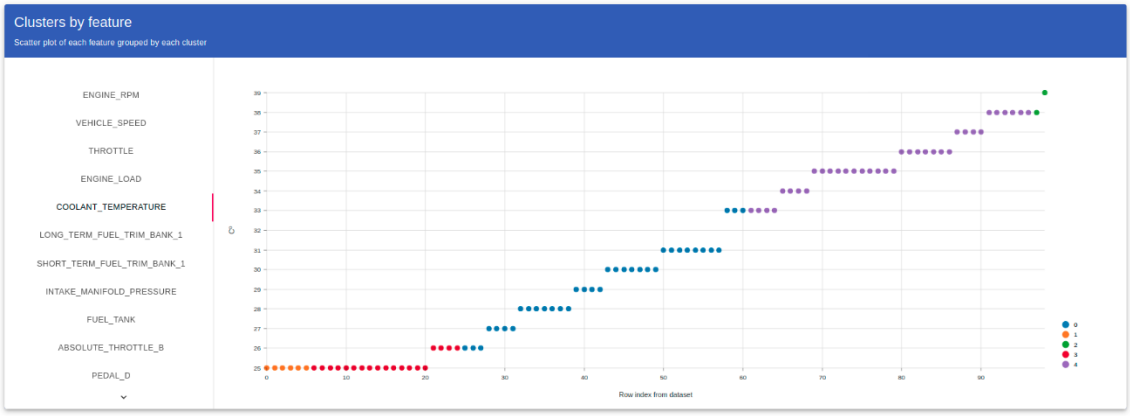


Figura 7.12 Diagrama de puntos con los datos de cada sensor categorizado por tipo de clúster.

7.4.5 Dificultades encontradas

A lo largo del desarrollo del cliente se han encontrado las siguientes dificultades:

- **TypeScript:** añadir TypeScript al proyecto, aunque nos aporta muchísimas ventajas y nos previene muchísimos errores, aporta un grado de dificultad a la hora de implementar componentes. Para empezar, implica una carga de trabajo mayor puesto que hay que definir todas las interfaces y crear los componentes atendiendo a ellas. También, no todas las librerías que existen para ReactJS tienen soporte para TypeScript, por lo que estaremos trabajando a veces a ciegas. Finalmente, para elaborar estructuras más complejas como mapas (en TypeScript se ha usado el tipo *Record*), contiene muchas limitaciones a la hora de trabajar con colecciones de datos. Por ello, añadir TypeScript al proyecto se ha traducido también en destinar mucho más tiempo para la implementación y lidiar con muchos errores.

- **Nivo:** usar Nivo junto con TypeScript ha aumentado la complejidad en el mapeo de datos. Las estructuras que se reciben desde la API no se pueden aplicar directamente a los gráficos. Nivo implementa su propia estructura y en la documentación no se explica explícitamente cómo formar los datos. El hecho de mencionar TypeScript en este apartado hace referencia a la creación de mapas y otras complejas colecciones para poder enviarlas a los componentes de Nivo.

7.4.6 Pruebas realizadas

Junto con la implementación del cliente, es necesario realizar pruebas unitarias para comprobar el buen funcionamiento del código. En este caso, ReactJS incluye Jest como *framework* para hacer *tests.*

La convención que suele seguir la comunidad e interpreta Jest es crear dentro de la carpeta *src/* una carpeta llamada *__tests__ y* dentro de ella, de forma jerarquizada o no añadir los *tests*.

Para crear un *test* simplemente hay que crear un archivo TypeScript (*.ts*) y utilizar las etiquetas que ofrece Jest para organizarlo. En este caso, una convención muy usada por la comunidad es la representada en la Figura 7.13.

```
describe("Bag", () => {
  describe("when you try to add a new element to the bag entity", () => {
    it("should add the element to the bag", () => {});
  });
});
```

Figura 7.13 Ejemplo de la estructura de un *test* usando Jest.

Finalmente, una vez se han desarrollado los *tests,* basta con ejecutar *yarn test* en la raíz del proyecto para que se ejecuten todos los *tests.*

7.6 Implementación del servicio de IA

Finalmente, tras haber implementado el cliente web y la API, se ha desarrollado el módulo de inteligencia artificial mediante Python. Para ello se ha creado una aplicación mediante Flask que permite aplicar cada algoritmo de IA sobre *datasets* de entrada.

A continuación, se detallan que tecnologías se han usado, la estructura que se ha seguido, las rutas definidas para Flask, la implementación de cada uno de los tres algoritmos, las dificultades encontradas en su desarrollo y las pruebas realizadas.

7.6.1 Tecnologías

Para desarrollar el servicio de inteligencia artificial se ha utilizado Python como lenguaje base. Para poder servir los modelos de inteligencia artificial al sistema, concretamente a la API que es la encargada de gestionar dichas peticiones, se ha utilizado Flask para poder montar una aplicación estableciendo las rutas correspondientes para interaccionar con los distintos modelos.

En cuanto a la implementación de los modelos de *Machine Learning,* se ha utilizado scikit-learn como librería la cual cuenta con multitud de recursos para poder aplicar los diferentes algoritmos que se han elegido.

7.6.2 Estructura

Una vez vistas las tecnologías usadas, a continuación, se describe la estructura que se ha seguido en la implementación de este servicio:

- ***ai/:*** este directorio se corresponde con el paquete de IA, en el que se encuentra un módulo para cada algoritmo, es decir, un archivo *k_means.py*, *svm.py* y *lstm.py*, en los cuales se implementan los distintos algoritmos de *Machine Learning.*

- ***app.py:*** este archivo se encarga de iniciar el servidor y en él se configuran las rutas que soporta el servicio, así como la llamada a los módulos respectivos para aplicar los modelos de IA.

7.6.3 Rutas

Para poder servir las clasificaciones y predicciones al exterior, es necesario definir una serie de rutas con las que va a interactuar nuestra API:

Ruta	/classify
Método HTTP	POST
Descripción	Aplica una clasificación por medio de *k-means* y SVM sobre el *dataset* enviado
Request	• *components-number:* número de componentes principales que calcular • *clusters-number:* número de categorías que identificar
***Form* (formulario Post)**	• *csv: dataset* en formato CSV

Tabla 7.10 Definición de ruta POST /classify.

Ruta	/svm
Método HTTP	POST
Descripción	Aplica una clasificación sobre un *dataset* dado aplicando el modelo de SVM entrenado
***Form* (formulario Post)**	• *csv: dataset* en formato CSV

Tabla 7.11 Definición de ruta POST /svm.

Ruta	/predict
Método HTTP	POST
Descripción	Realiza una predicción sobre un sensor determinado utilizando el *dataset* original como entrenamiento
Request	• *feature:* característica elegida para predecir • *epochs:* número de épocas elegidas para la red neuronal • *predictions-feature-type: parámetro que determinada si se usan todos los sensores, uno o PCA para el entrenamiento* • *pc-number:* en el caso de elegir PCA, el número de componentes principales

***Form* (formulario Post)**	• *csv: dataset* en formato CSV

Tabla 7.12 Definición de ruta POST /predict.

7.6.4 *k-means*

El primer modelo aplicado ha sido *k-means*, en el que, a partir de un *dataset* de entrada sin etiquetar, se quiere determinar un número de categorías en las que se puedan clasificar los datos y poder encontrar un sentido a los mismos.

Tras normalizar los datos, el primer paso ha sido aplicar PCA sobre todos los datos para así poder reducir la dimensionalidad de los mismos y obviar datos redundantes. Para ello se crea una instancia de PCA asignando el número de componentes principales que se quieren determinar. Una vez se obtienen las componentes principales, se forma un *dataset* con ellas y se aplica *k-means* sobre él. Finalmente se forma un *dataset* con las componentes principales y los clústeres asociados a cada entrada.

Para mostrar gráficamente las componentes principales y la clasificación realizada, se crea un gráfico con las dos componentes principales utilizando como leyenda los clústeres obtenidos. También se crea otro gráfico que muestra la participación de cada característica para cada componente principal.

Alternativamente, para poder refinar este modelo también se forma un gráfico con la varianza acumulada tras aplicar PCA sobre nuestros datos. Esto nos puede permitir determinar cuántas componentes principales son necesarias para conseguir el porcentaje de representación deseado (en este caso se ha fijado un 90%).

Por otro lado, para determinar cuál es el número de clústeres óptimo para categorizar, se ha realizado la distancia media cuadrática de cada punto respecto a su centroide a lo largo de un rango de 21 clústeres. Para determinar el número apropiado, se ha hecho uso de una librería (*KneeLocator*) que aplica el método del codo sobre dichos datos.

El método del codo (*Elbow Method)* utiliza los valores de inercia obtenidos tras aplicar *k-means* a un número de clústeres (de 1 a N clústeres) siendo N en este caso 21, siendo la inercia la suma de las distancias al cuadrado de cada elemento del clúster a su centroide. La fórmula de este método se corresponde con la Figura 7.14.

$$Inercia = \sum_{i=0}^{N} \|x_i - \mu\|^2$$

Figura 7.14 Fórmula para calcular la inercia por medio de la técnica del codo.

Finalmente se obtiene un gráfico que nos muestra el punto (el número de clústeres) más apropiado. Puesto que en el cliente web se puede elegir el número de clústeres por

medio de un formulario, se puede volver a realizar una clasificación atendiendo a este gráfico.

Finalmente, a modo de información y a partir de la aplicación de PCA, se obtienen las características más representativas para cada componente principal.

7.6.5 SVM

Una vez realizado *k-means* sobre los datos, ya obtenemos un *dataset* etiquetado y es cuando se puede aplicar SVM sobre los mismos, ya que al ser un método supervisado se precisa de un *dataset* etiquetado. Se pretende entrenar un modelo de SVM capaz de clasificar nuevos datos en las categorías o etiquetas que se han definido.

En primer lugar, sobre el *dataset,* se realiza una división entre datos de entrenamiento y de test por medio de *train_test_split()*, una función de Sklearn que divide el *dataset* en datos de entrenamiento y test aleatorios de forma eficiente.

En segundo lugar, se define la función que se va a utilizar como *kernel* en el modelo SVM. La función *kernel* se encarga de transformar los datos de entrada a su formato apropiado para ser usado por el algoritmo. Existen diversas: linear, no linear, RBF... En la implementación de SVM se ha utilizado RBF.

Una vez configurado el *kernel* con sus parámetros *gamma* y *C,* se crea la instancia de SVM. Una vez se ha creado, es paso de entrenar el modelo con los datos de entrenamiento previamente obtenidos.

Finalmente, una vez se ha entrenado, es hora de realizar clasificaciones. Para ello se toman los datos de test que se han obtenido previamente y se realiza una clasificación sobre ellos, obteniendo un listado con las categorías que ha determinado el modelo para cada uno de ellos. Para mostrar el resultado al usuario, se ha creado un gráfico en el que se muestran los diferentes hiperplanos creados, los vectores soporte y los datos clasificados en las diferentes categorías.

7.6.6 Red neuronal LSTM

Paralelamente al desarrollo de los modelos de clasificación, se ha implementado una red neuronal LSTM para realizar predicciones.

En primer lugar, se ha de procesar el *dataset* dependiendo de cuántas características (series) y cómo las queremos tener en cuenta. Si se decide realizar la predicción utilizando únicamente como datos de entrenamiento los de la característica que se desea predecir, hay que eliminar el resto de las características del *dataset*. Si por otro lado se pretende aplicar PCA, primero se aplica PCA sobre los datos y se le adjunta la columna de la característica que se pretende predecir. En el caso de querer tener en cuenta todas las características del *dataset,* no se requiere ningún paso especial.

A continuación, se procede a la normalización de los datos utilizando *MinMaxScaler* que establece el rango de la normalización utilizando como valor máximo y mínimo el respectivo de dicha característica.

Una vez se tienen los datos, se establecen cuantas series temporales anteriores (t-3, t-2, t-1, t) se quieren tener en cuenta para realizar las predicciones. Por lo tanto, se extiende el *dataset* de X columnas a X * t columnas más la columna de la característica que se pretende predecir.

A continuación, se divide ya el *dataset* en datos de entrenamiento y de test y se transforman adecuadamente para ser aceptados por la red neuronal LSTM.

Una vez los datos han sido totalmente procesados y listos para entrenar, primero se ha de configurar la red neuronal. Se configuran las distintas capas que se pretenden agregar y se compila el modelo utilizando como *loss* el Error Cuadrático Medio (MSE) y como optimizador *Adam*.

Una vez se ha compilado el modelo, es paso de comenzar el entrenamiento. Para ello primero se necesitan configurar una serie de parámetros: el número de épocas (número de veces que se procesa el *dataset*), el *batch size* (número de datos de entrenamiento en los que se divide el *dataset* para que sea procesado de manera más fácil) y la cantidad de datos usados para validar el modelo. Tras ello, se procede al entrenamiento con los datos de entrenamiento previamente obtenidos.

Una vez se ha finalizado el entrenamiento, se genera un gráfico con la curva de aprendizaje donde se puede observar el error entre los datos reales y los que se han precedido, acompañado de calcular el error cuadrático medio. Dichos datos son muy útiles y pueden ayudar a determinar si la red neuronal ha estimado correctamente o presenta mucho error.

Finalmente, al tener entrenado el modelo, el siguiente paso consiste en visualizar la predicción que se ha realizado. Para ello, en primer lugar, hay que desnormalizar los datos, y posteriormente, se genera un gráfico donde se representa tanto el valor real como la predicción a lo largo del tiempo.

7.6.7 Dificultades encontradas

A lo largo del desarrollo del servicio se han encontrado diversas dificultades referentes a la aplicación de los métodos de *Machine Learning* puesto que integrar Flask ha sido fácil y cómodo (es una de sus características principales).

Por ello las dificultades se reducen a la aplicación de cada algoritmo, que en primer lugar conlleva un largo proceso de familiarización con su funcionamiento teórico, y, en segundo lugar, aplicarlo por medio de programación.

7.6.8 Pruebas

Para comprobar el correcto funcionamiento del código se han realizado pruebas unitarias por medio de la librería *unittest* que incluye Python. Para ello, se ha creado

junto con cada archivo testeado (siguiendo las convenciones de la comunidad) un archivo con el mismo nombre con el prefijo *test_*: si por ejemplo se pretenden hacer pruebas sobre el módulo *common.py*, el archivo se llamará *test_common.py*.

La estructura del *test* se puede reflejar en la Figura 7.15: creando una clase haciendo referencia al módulo y creando funciones para cada función que se quiere *testear*. Dentro de cada una se pueden incluir diferentes comprobaciones, aunque en este ejemplo se ha reducido por simplicidad.

```
class TestCommon(unittest.TestCase):
    def test_generate_pc_columns_names(self):
        self.assertAlmostEqual(generate_pc_columns_names(2), ['pc1', 'pc2'])
```

Figura 7.15 Ejemplo de prueba unitaria usando *unittest*.

8

Resultados y experimentos

Tras haber explicado cómo se ha desarrollado todo el proyecto, en este capítulo, mostraremos, a nivel de ejemplo, los escenarios de uso más habituales del sistema desarrollado.

8.1 Clasificación sobre un *dataset* sin etiquetar

Todos los *datasets* que se han usado como fuente de datos no están etiquetados. El primer paso para poder gestionarlos es aplicar *k-means* sobre los mismos para que pueda clasificarlos y poder encontrar un sentido sobre los mismos.

Para ello, este experimento se va a realizar sobre un *dataset* que se corresponde con un recorrido en el que inicialmente el vehículo se encuentra estacionado, luego realiza una distancia por un campus a baja velocidad y finalmente circula por la carretera a más velocidad. En total cuenta con 3000 registros (filas en el *dataset).*

Tras haber añadido el *dataset* al sistema, en un primer paso se realiza una clasificación inicial seleccionando 3 clústeres y se obtiene, entre otros, el gráfico reflejado en la Figura 8.1. Se corresponde con un análisis para encontrar el número óptimo de clústeres (o categorías) para el *dataset* dado por medio de la técnica de WCSS. Podemos comprobar como aplicando la técnica del codo se indica que el número óptimo de clústeres es 5. Al diferir del número inicial de clústeres que se han elegido, se procede a realizar una segunda clasificación seleccionando 5 clústeres en el formulario.

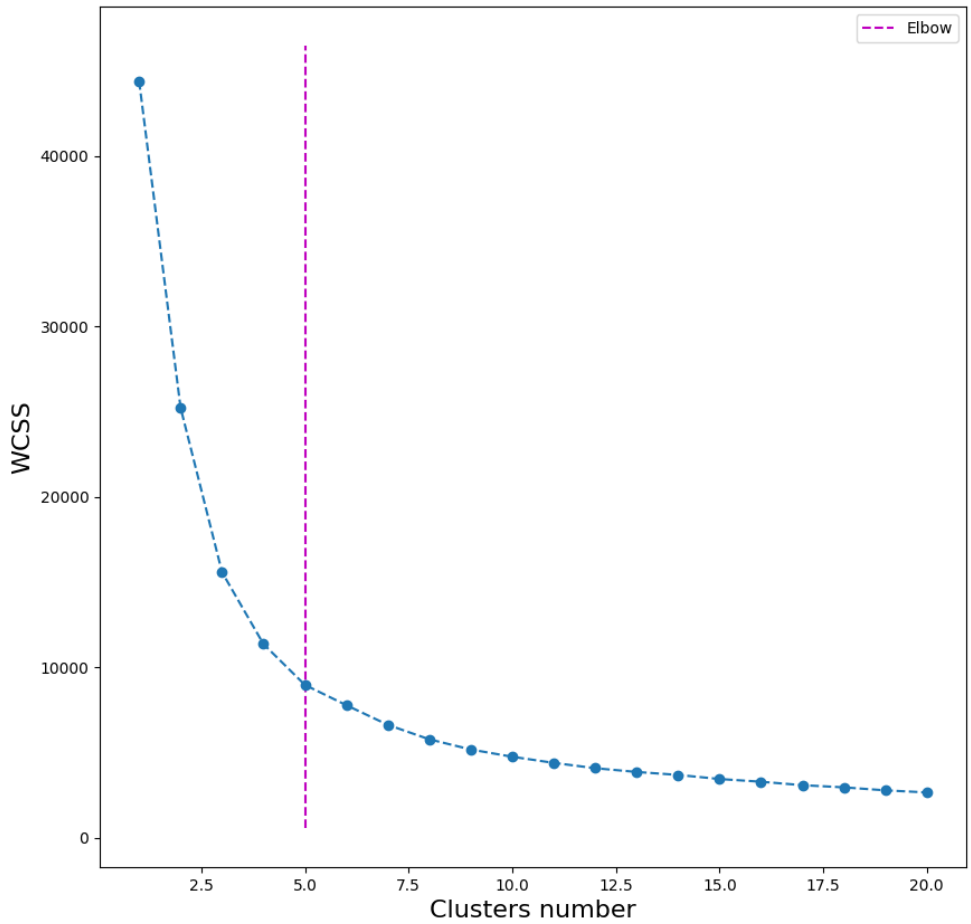


Figura 8.1 WCSS y técnica del codo aplicada sobre el dataset.

Tras obtener un resultado de clasificación por *k-means* con el número apropiado de clústeres, podemos ya analizar los datos obtenidos. En primer lugar se genera un gráfico que representa las dos primeras componentes principales tras aplicar PCA sobre el *dataset* junto con los datos ya etiquetados. Se puede observar cómo en la Figura 8.2 aparecen los 5 clústeres definidos bien diferenciados, por lo tanto podemos concluir que el método *k-means* ha encontrado de forma adecuada las categorías que se esperaban.

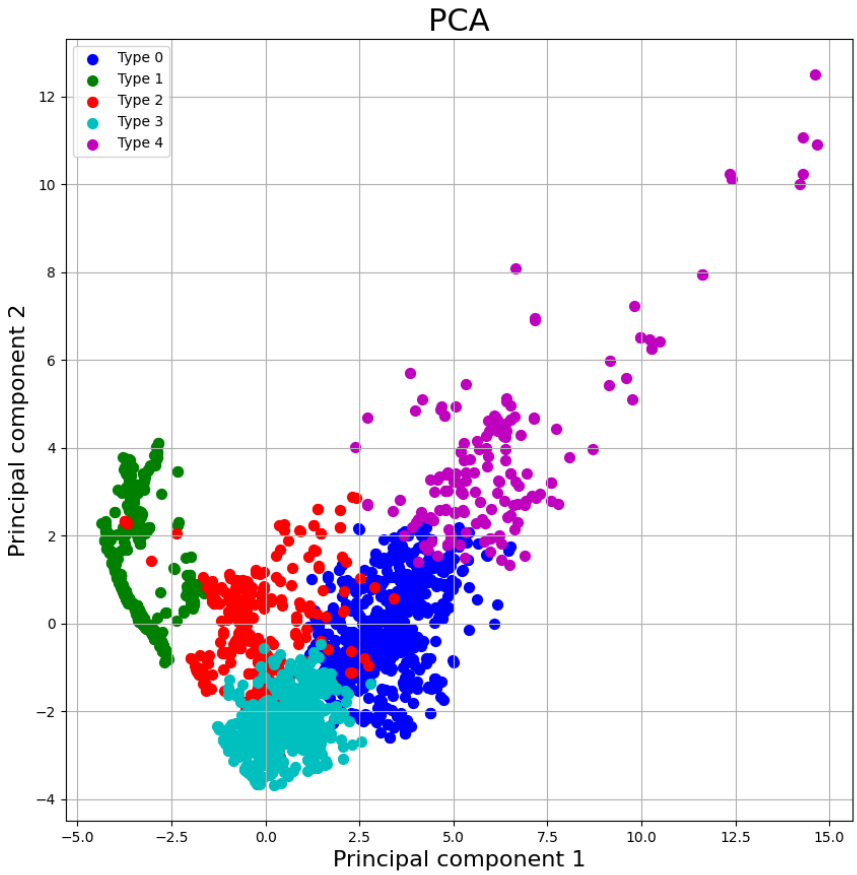


Figura 8.2 Gráfico de las dos primeras componentes principales junto con la clasificación por *k-means*.

Alternativamente, si se quiere consultar la influencia que ha tenido cada característica (sensor) tras aplicar PCA, se puede observar en la Figura 8.3 cómo cada una de las características tiene mayor o peor preso sobre cada uno de los componentes principales. Por ejemplo, para el componente principal 1, se observa que el *PEDAL_D* y *PEDAL_E* han tenido más influencia, lo cual indica que son sensores que realmente tienen una importancia en el resultado final y no son prescindibles.

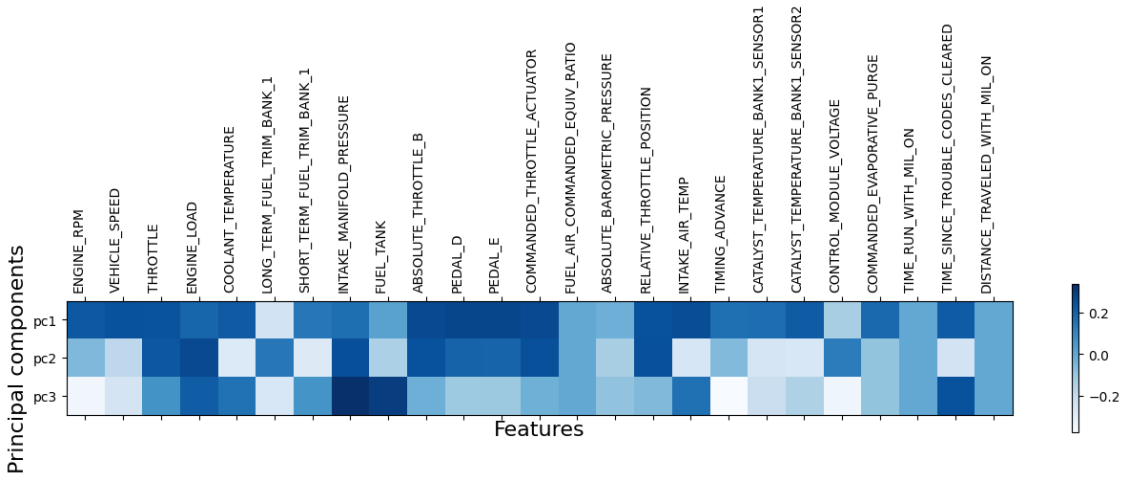


Figura 8.3 Gráfica de la influencia de cada característica respecto a las componentes principales.

Finalmente, si se quiere observar en mayor detalle la relación que existe entre cada clúster o categoría de la clasificación, mediante la herramienta que se ha desarrollado para mostrar cada característica o sensor respecto a cada categoría, se puede observar como en la Figura 8.4, que se corresponde con la velocidad del vehículo, se puede distinguir claramente una categoría (los primeros mil registros) que conforma una categoría claramente definida. Este razonamiento tiene bastante sentido puesto que dichos registros corresponden al vehículo estacionado sin movimiento. Por otro lado, el *dataset* no ha conseguido distinguir de forma muy precisa la diferencia entre un recorrido a menor velocidad y otro en carretera. Este resultado puede deberse a que los *datasets* obtenidos de la fuente de datos no sean lo suficientemente precisos y sean recorridos similares.

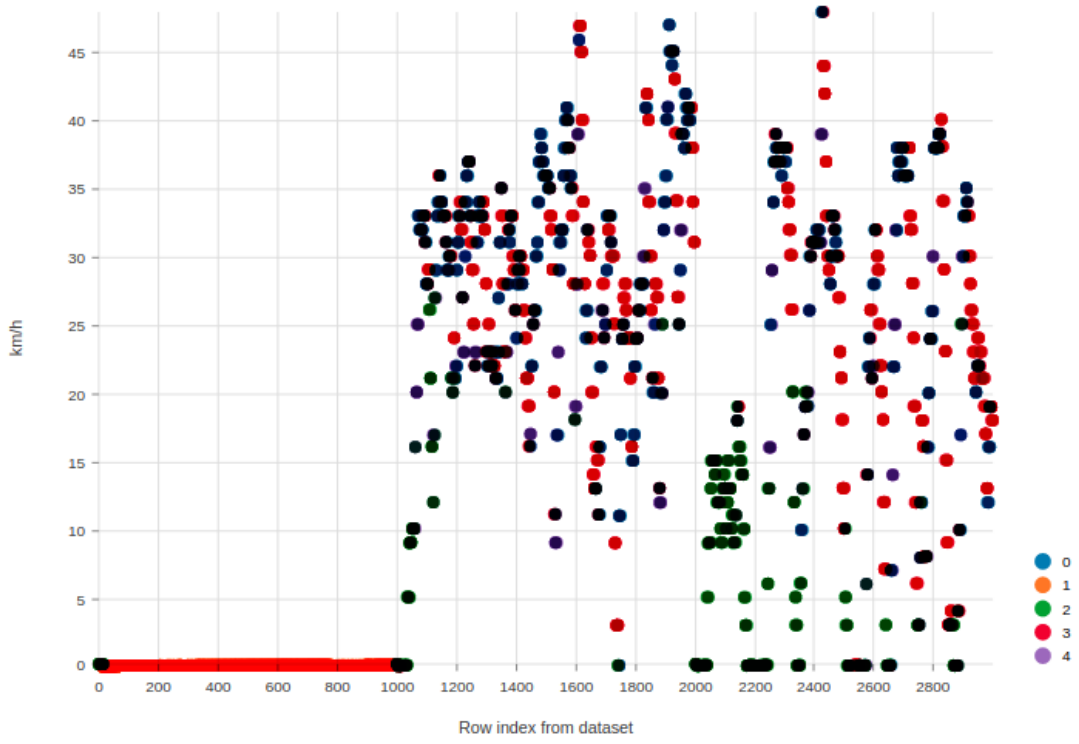


Figura 8.4 Velocidad del automóvil clasificada por clúster.

8.2 Clasificación de un *dataset* usando SVM

Tras haber podido clasificar un *dataset* sin etiquetar y encontrar grupos claramente definidos por medio de *k-means*, ahora es posible entrenar el modelo de clasificación SVM desarrollado y poder clasificar nuevos datos de entrada.

El planteamiento del experimento va a ser el siguiente: reutilizando el *dataset* usado en el experimento anterior que ya está clasificado por medio de *k-means*, se va a entrenar el modelo de SVM y, posteriormente, mediante la herramienta de clasificación, se va a intentar predecir un *dataset* que se corresponde con las últimas diez filas del *dataset* inicial. De esta manera se puede comparar las clasificaciones iniciales con las nuevas y ver si el modelo ha clasificado bien.

Para empezar, se entrena el modelo con el *dataset* ya clasificado y se obtiene el gráfico de la Figura 8.5 que muestra las dos primeras componentes principales de aplicar PCA junto con los hiperplanos y vectores soporte generados por SVM y cada categoría a modo de leyenda.

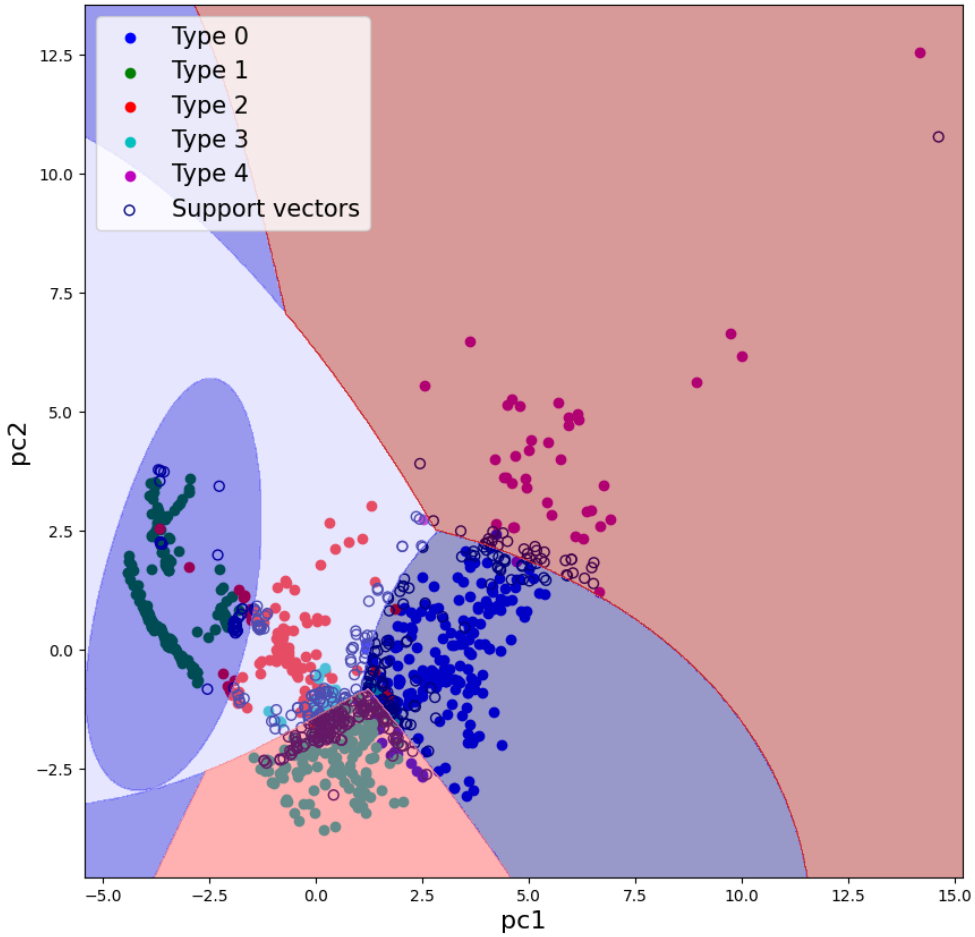


Figura 8.5 Gráfica de SVM tras entrenar el modelo.

Se puede observar que claramente el resultado es similar al de *k-means* puesto que se ha hecho a partir de su etiquetado y los hiperplanos son adecuados.

A continuación, mediante la herramienta de clasificación que se encuentra en el panel de SVM, se procede a añadir otro *dataset* que se corresponde con las últimas diez filas del original. Tras haber realizado la nueva clasificación, se obtiene el mismo *dataset* con una nueva columna de la clasificación por medio de SVM. Las etiquetas (categorías) de las ultimas diez entradas del *dataset* original se corresponden con [0,3,3,3,3,3,3,3,3,3] respectivamente. Por otro lado, la clasificación para el *dataset* que se ha analizado es [1,3,3,3,3,1,3,3,3,3] respectivamente. Por lo tanto, podemos concluir que el modelo de SVM clasifica adecuadamente puesto que, de un total de diez entradas, ha clasificado bien ocho.

8.3 Predicción sobre un sensor por medio de LSTM

Tras haber finalizado los experimentos con el módulo de clasificación, se comienza el experimento sobre el módulo de predicción. Para ello, haciendo uso de la red neuronal LSTM diseñada, se entrena sobre un *dataset* para poder predecir características (sensores).

Para este experimento se va a utilizar el mismo *dataset* que el resto de los experimentos. La primera prueba que se realiza es prediciendo un sensor usando como únicamente fuente de datos dicha columna del *dataset*, es decir, sin tener en cuenta el resto de las variables. Se ha decidido predecir el nivel de combustible del vehículo para poder ofrecer un caso práctico lo más cercano a la población. La red neuronal se va a configurar con 150 épocas, y tras aplicar la predicción, se ha obtenido un error cuadrático medio (RMSE) de 0.05044 y ha tardado en total 137 segundos. Los resultados son bastante buenos puesto que el error es mínimo. Para fijarnos en más detalle sobre el error, a continuación, en la Figura 8.6 se puede observar una gráfica con la curva de aprendizaje de la red neuronal dónde se representa como *train* los datos entrenados y como *test* los datos que se han predicho. A su vez, se puede ver como a partir de la época 50 el error decrece poco a poco, pero permanece prácticamente constante. Se han realizado más pruebas a partir de 150 épocas, pero el error aumenta y no se consiguen resultados tan buenos. Finalmente, obtenemos la predicción reflejada en la Figura 8.8, dónde las predicciones han conseguido un nivel de precisión bastante bueno.

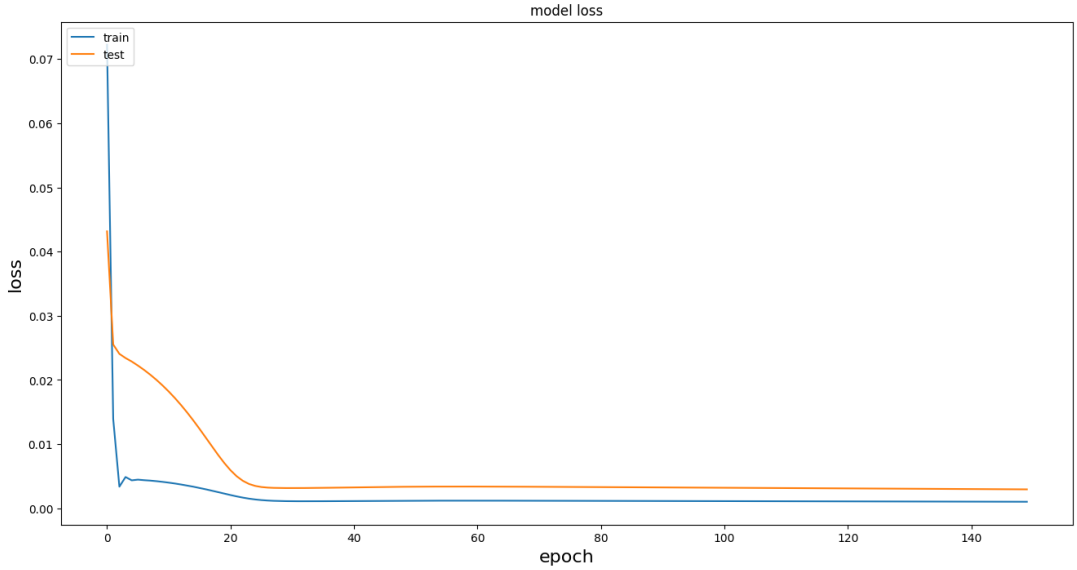


Figura 8.7 Curva de aprendizaje usando una sola característica.

Figura 8.8 Predición del nivel de combustible del vehículo usando una sola característica.

Tras haber realizado la predicción sobre el nivel de combustible teniendo en cuenta únicamente dicha columna del *dataset*, un escenario interesante podría ser investigar si teniendo el resto de características (sensores) en cuenta en la predicción, se pueden obtener predicciones más preceisas. Para ello, se ha vuelto a realizar la predicción con la misma configuración, únicamente utilizando la totalidad del *dataset*. No obstante, nos encontramos con que los resultados empeoran considerablemente, aumentando tanto el RMSE de 0.30180 como el tiempo de entrenamiento, 200 segundos. En cuanto a las gráficas generadas (figuras 8.9 y 8.10) se puede observar como claramente los resultados no son nada precisos y nos encontramos ante un caso de *underfitting*.

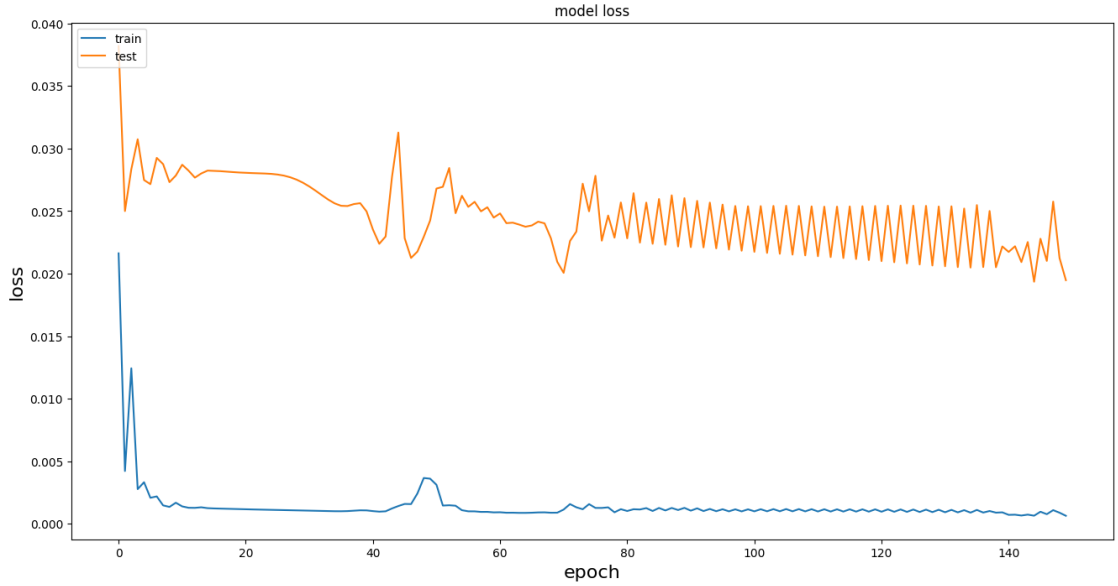


Figura 8.9 Curva de aprendizaje usando todas las características.

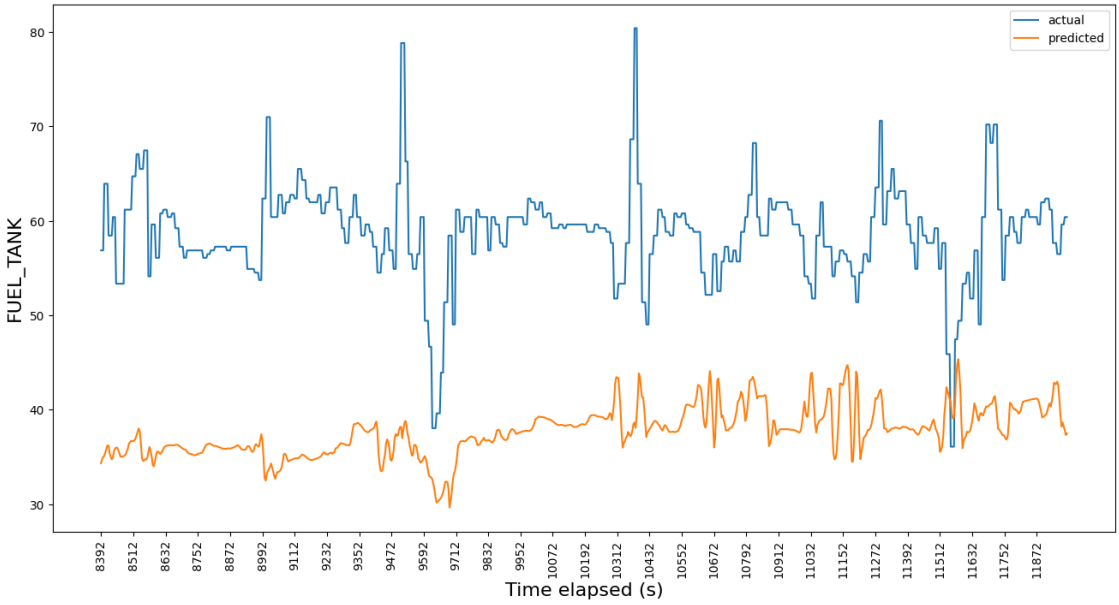


Figura 8.10 Predición del nivel de combustible del vehículo usando una sola característica.

Aunque se hayan obtenido estos resultados tan desfavorables, aun hay un tercer escenario que se puede investigar que consiste en aplicar PCA sobre las características para eliminar información redundante y reducir considerablemente las entradas de la red neuronal para conseguir predicciones más precisas que en el caso de utilizar directamente todas las características. Para ello se va a volver a realizar una predicción sobre el nivel de combustible, utilizando el mismo número de épocas y esta vez aplicando PCA antes de entrenar la red neuronal. Tras aplicar la predicción, se obtienen resultados mejores que en el escenario de usar todas las características pero peores que utilizar solamente la característica que se pretende predecir: se obtiene un RMSE de 0.16319 y un tiempo de entrenamiento de 140 segundos. Los resultados gráficos se pueden observar en las figuras 8.11 y 8.12. Se ha investigado si aplicando más épocas se

consigue reducir el error pero en este caso permanece constante y no se consigue ninguna mejora.

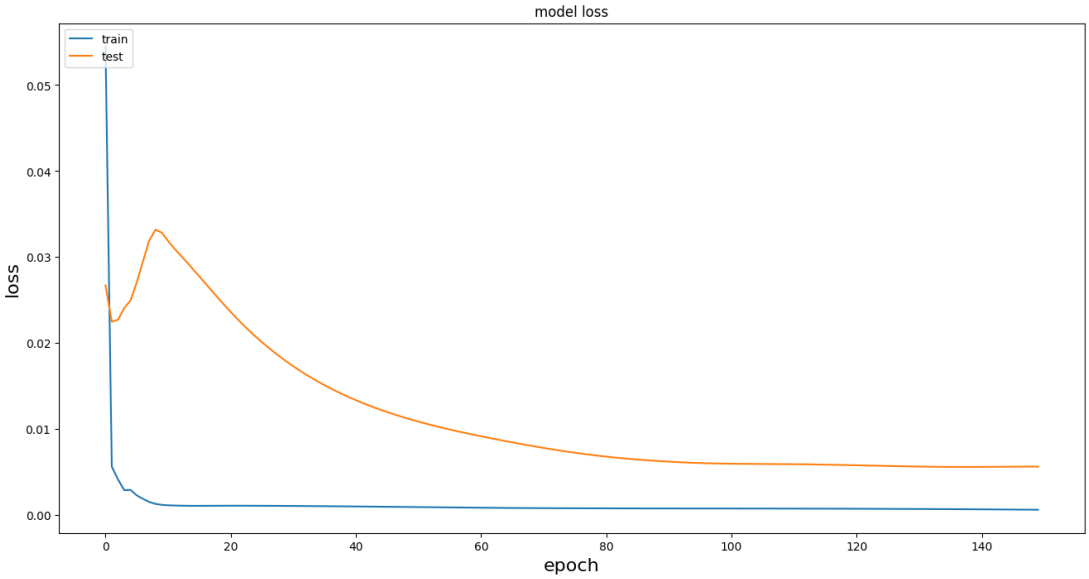


Figura 8.11 Curva de aprendizaje aplicando PCA.

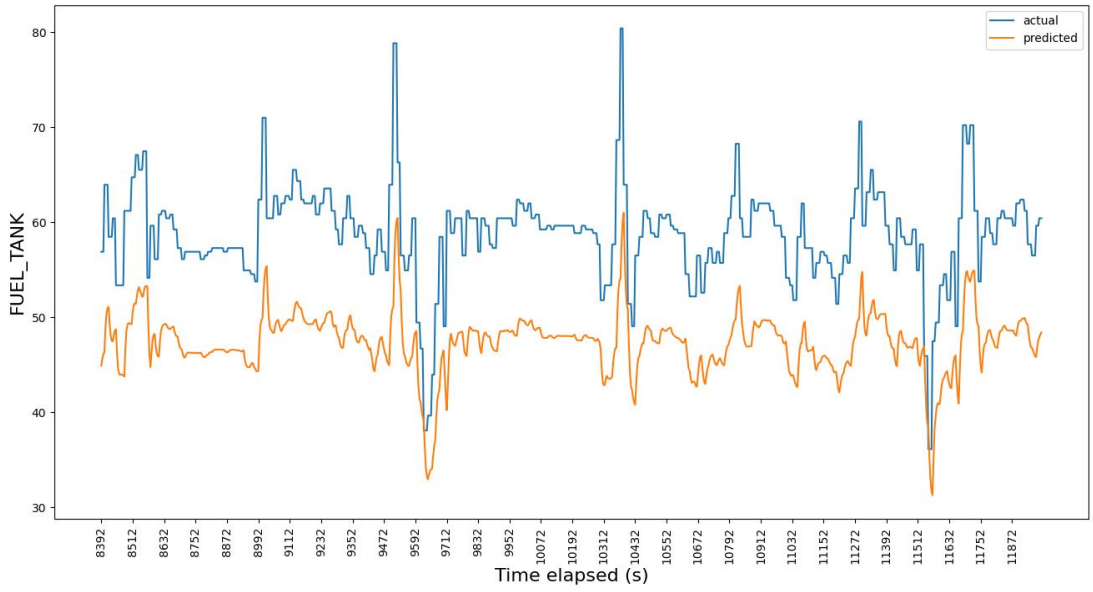


Figura 8.12 Predición del nivel de combustible del vehículo aplicando PCA.

Como conclusión final tras este análisis investigando la relevancia que tiene usar más o menos cantidad de datos, se concluye que aunque aplicando PCA el tiempo de entrenamiento es prácticamente igual que usando una única característica, el error es tres veces mayor y no se consigue ninguna ventaja. A su vez se suma que usando todas las características el tiempo de entrenamiento aumenta considerablemente y el error se dispara hasta seis veces más que usar una única característica. Por lo tanto, no se

obtiene ninguna mejora teniendo en cuenta el resto de datos pues aumenta la redundancia en los mismos y la precisión en las predicciones disminuye.

9

Conclusiones y líneas futuras

9.1 Conclusiones

El desarrollo del TFG ha tenido dos principales objetivos: por un lado, de carácter técnico, es decir, aprender nuevas tecnologías y poder aplicarlas para formar el sistema; y, por otro lado, de carácter científico, al aprender los conceptos básicos del *Machine Learning* y poder crear modelos de Inteligencia Artificial capaces de integrarse con el sistema que se ha montado.

9.1.1 Tecnologías

Respecto al primer apartado de aprender nuevas tecnologías y aplicarlas se ha cumplido con totalidad: se han aprendido lenguajes de programación y *frameworks* novedosos como ReactJS, Material UI, Golang, Python, Flask... que, aunque en el transcurso del grado no se incluyen en los planes de estudio, los conceptos adquiridos durante estos años sobre programación, base de datos, desarrollo web o modelado han servido de base para su aplicación con total éxito.

En general, el resultado de haber utilizado diferentes tecnologías para cada servicio ha sido bueno, puesto que se ha podido aprovechar las ventajas de cada uno. Usar Go para desarrollar la API ha permitido desarrollarla de manera más rápida y siguiendo una estructura bastante clara. A parte, al haber añadido un ORM para la gestión de la base de datos ha ahorrado bastante trabajo para realizar consultas SQL puesto que en este contexto no se requería gran complejidad en ellas y utilizar esta herramienta nos evita rehacer el mismo código que ya nos genera el ORM. Por otro lado, utilizar ReactJS para desarrollar el cliente web junto con Material UI como *framework* ha resultado en una

buena interfaz gráfica sin necesidad de dedicar tiempo a crear elementos básicos como botones o tablas puesto que ya lo ofrece Material UI. Además, cuenta con una documentación bastante detallada para consultar cualquier duda en su integración. Finalmente, usar Python para desarrollar los servicios de *Machine Learning* ha sido un total acierto: cuenta con multitud de librerías matemáticas y de *Machine Learning* que han ayudado considerablemente en la aplicación de los diferentes algoritmos. A parte, Python cuenta con una comunidad bastante extendida en todo el mundo, por lo que se puede acceder a numerosa documentación y tutoriales para resolver cualquier duda.

En cuanto a las desventajas que se han encontrado, se puede decir que en concreto sobre ninguna se ha notado algún problema considerable. No obstante, si hablamos en el sentido general, el haber utilizado diferentes tecnologías ha hecho un tanto más complejo la comunicación entre ellas, sobre todo entre la API y el servicio de IA.

9.1.2 Técnicas de Inteligencia Artificial

En cuanto a la aplicación de *Machine Learning*, se ha conseguido poder tener un primer contacto con la Inteligencia Artificial: desde conocer las diferentes categorías que existen, su principal uso y su aplicación sobre sistemas reales. Ha aportado un grado de dificultad al desarrollo del proyecto, pero ha ayudado considerablemente a aumentar su utilidad y a poder afianzar los principales conceptos de cada algoritmo aplicado.

En primer lugar, aunque la aplicación de PCA ha estado presente como parte del procesamiento de datos previos a aplicar el resto de los algoritmos de Inteligencia Artificial, ha sido muy útil en *k-means* y SVM para que sean más eficientes y los resultados sean más precisos al eliminar información redundante. No obstante, respecto a su aplicación en la red neuronal no ha resultado de gran utilidad puesto que, aunque aumenta la eficiencia de la red respecto a utilizar la totalidad de los datos, los resultados son ligeramente peores que aplicar la predicción utilizando una sola característica.

En segundo lugar, aplicar *k-means* ha sido de gran utilidad puesto que los *datasets* que se estaban usando como fuente de datos no estaban etiquetados, por lo tanto, se necesitaba una herramienta que los etiquetara y poder encontrarle un sentido. A parte, nos ha servido para posteriormente usar SVM. No obstante, tras realizar algunas aplicaciones concretas, no ha conseguido diferenciar del todo algunos recorridos distintos.

En tercer lugar, la aplicación de SVM nos ha sido realmente útil para permitir clasificar nuevos *datasets* de entrada con buena precisión. Por otro lado, esta función ya nos la permite *k-means* y, además, en este caso, no podríamos haber utilizado SVM sin haber aplicado *k-means* previamente por lo que en esta dependencia se puede encontrar cierta desventaja en su uso.

Finalmente, el desarrollo de la red neuronal por medio de LSTM ha ofrecido resultados bastante buenos y también con una utilidad más cercana al contexto de los vehículos, puesto que se han obtenido predicciones sobre sensores concretos del vehículo: nivel de combustible, temperatura... No obstante, por limitaciones de *hardware no* se ha

podido trabajar con *datasets* mucho más grandes y aumentar la complejidad de la red neuronal.

9.1.3 Utilidad de la herramienta desarrollada

Finalmente, tras haber analizado tanto el contexto de tecnologías como la integración de modelos de Inteligencia Artificial, se ha desarrollado una herramienta bastante útil para poder investigar y analizar los sensores de un vehículo. Permite encontrar patrones a lo largo de un recorrido mediante el módulo de clasificación y de esta manera, se le puede dar un sentido a los datos y determinar, por ejemplo, patrones de conducción.

También resulta bastante útil para poder realizar predicciones sobre algunos casos concretos de sensores: el consumo del vehículo o la temperatura de ciertos componentes. Esto nos puede ayudar a determinar anomalías o problemas que pueda sufrir el coche en un futuro. No obstante, para que sean más precisas y se puedan extender más en el tiempo la herramienta necesita mejorar la infraestructura en la que se utiliza como previamente se ha comentado en el capítulo anterior: sería bastante interesante poder mejorar el módulo de predicción accediendo a recursos de *hardware* más potentes, por ejemplo, los que ofrecen algunas empresas como Google o Amazon en sus servidores por medio de *GPUs*.

9.2 Líneas futuras

Aunque el sistema que se ha desarrollado es totalmente funcional y cumple con los objetivos definidos inicialmente, se puede elaborar un listado de complementos que podrían mejorar el sistema y aportar más utilidades:

- **Desarrollar un servicio para recolectar datos usando un lector OBD-2:** la fuente de datos utilizada para el proyecto ha sido obtenida a través de un *dataset* publicado en Kaggle. Se podría elaborar un servicio, que conectado a un dispositivo electrónico con el puerto OBD-2 de un vehículo, pueda ir recolectando datos sobre los sensores en un recorrido y poder obtener dichos *datasets* para añadirlos y procesarlos en la aplicación creada. De esta manera, se podrían hacer pruebas sobre diferentes vehículos.

- **Añadir Tensorflow Serving y despliegue en *cloud***: se podría añadir Tensorflow Serving para el servicio de *Machine Learning*, especialmente para la red neuronal, junto con un despliegue del sistema en la nube. Por ejemplo, se podría adquirir un servidor con suficiente memoria y potencia (se podría hacer uso de GPUs), que junto con Tensorflow Serving, ayudase a mejorar la eficiencia de nuestros modelos de inteligencia artificial y poder conseguir resultados en menor tiempo y más certeros.

- **Incluir histórico para las clasificaciones y predicciones:** actualmente los datos se sobrescriben en cada nueva operación. Sería interesante montar

un panel en cada apartado para poder gestionar el histórico con todas las clasificaciones o predicciones que se han realizado.

Referencias

[CS18] Scott Chacon y Ben Straub. “Pro Git.” Apress. 2018.

[D20] Docker Inc. https://docs.docker.com/compose/. 2020. Accedido: 10/09/2020.

[G16] Guttag, John V. “Introduction to Computation and Programming Using Python: With Application to Understanding Data”. Mitpress. 2016

[G20] Google. https://godoc.org/github.com/gin-gonic/gin. 2020. Accedido: 10/09/2020.

[H11] Marijn Haverbeke. “Eloquent JavaScript: A Modern Introduction to Programming.” Amazon. 2011.

[J20] Jinzhu. https://gorm.io/index.html. 2020. Accedido: 10/09/2020.

[M16] Mouat, Adrian, “Using Docker: Developing and Deploying Software with Containers”. O’Reilly. 2016.

[M19] Boris Cherny. “Programming TypeScript: Making Your JavaScript Applications Scale.” O’Reilly. 2019.

[M20] Microsoft Inc, “Visual Studio Code”, https://code.visualstudio.com. 2020. Accedido: 9/3/2020.

[P12] Rob Pike. "Go at Google: Language Design in the Service of Software Engineering". https://talks.golang.org/2012/splash.article. 2012. Accedido: 10/09/2020.

[P20a] Pallets. https://flask.palletsprojects.com/en/1.1.x/. 2020. Accedido: 10/09/2020.

[P20b] Postman, Inc. https://www.postman.com/. 2020. Accedido: 10/09/2020.

[S15] Sutherland, Jeff. “Scrum: The Art of Doing Twice the Work in Half the Time.” Random House Business. 2015.

[S20] Scikit-learn. https://scikit-learn.org/stable/. 2020. Accedido: 10/09/2020.

[T17] Oliver Theobald. “Machine Learning for Absolute Beginners.” Scatterplot Press. 2017.

[T20] The PostgreSQL Global Development Group. https://www.postgresql.org/. 2020. Accedido: 10/09/2020.

[W20] W3C. “HTML & CSS”. https://www.w3.org/standards/webdesign/htmlcss. 2020. Accedido: 10/09/2020.

[W20c] Wikimedia Foundation, Inc. https://en.wikipedia.org/wiki/GitHub. 2020. Accedido: 10/09/2020.

[Y20] Yarn. https://yarnpkg.com/. 2020. Accedido: 10/09/2020.

Apéndice A

Manual de Instalación

Requerimientos:

- Instalar Go
- Instalar ReactJS
- Instalar Git
- Instalar Python3
- Instalar Yarn
- Instalar Docker

Instalación de la base de datos:

Para la instalación de la base de datos no vamos a instalar nada en el sistema puesto que nos podemos servir de la portabilidad de Docker. De esta forma, con un simple comando se descargará la imagen de Postgres desde el *hub* de Docker y se inciará el servicio en el sistema.

1. Introducir el siguiente comando en el terminal:

docker run --name postgres -p 5432:5432 -e POSTGRES_PASSWORD=<contraseña> -d postgres

Instalación de la API:

Una vez se ha iniciado la base de datos, se puede comenzar a inicializar la API. Para ello, se compilará el código de Go y se activará el ejecutable resultante.

1. Abrir terminal.
2. Clonar repositorio (introducir comando en el terminal):

   git clone git@github.com:xFJA/intelligent-analysis-of-car-sensors-backend.git

3. Acceder mendiante el terminal a la raiz del repositorio.
4. Compilar aplicación (introducir comando en el terminal):

go build

5. Ejecutar aplicación (introducir comando en el terminal):

   ./intelligent-analysis-of-car-sensors-backend

Instalación del cliente web:

Tras iniciar la API, es momento de iniciar el cliente web. Para ello primero necesitaremos instalar todas las dependencias del servicio y posteriormente iniciar la aplicación mediante Yarn.

1. Abrir terminal.
2. Clonar repositorio (introducir comando en el terminal):

   git clone git@github.com:xFJA/intelligent-analysis-of-car-sensors-frontend.git

3. Acceder mendiante el terminal a la raiz del repositorio.
4. Instalar dependencias (introducir comando en el terminal):

   yarn install

5. Iniciar aplicación (introducir comando en el terminal):

   yarn start

Instalación del servicio de Machine Learning

Finalmente, solo queda iniciar el servicio de ML. Para ello instalaremos todas las dependencias que necesitamos y ejecutaremos la aplicación.

1. Abrir terminal.
2. Clonar repositorio (introducir comando en el terminal):

   git clone git@github.com:xFJA/intelligent-analysis-of-car-sensors-ai.git

3. Acceder mendiante el terminal a la raiz del repositorio.
4. Instalar dependencias (introducir comando en el terminal):

   pip install –r requirements.txt

5. Iniciar aplicación (introducir comando en el terminal):

   python app.py

Apéndice B

Manual de Usuario

Añadir *dataset*

Para añadir un nuevo *dataset* al sistema hay que navegar hacia la página de "Upload" haciendo “click” en la barra superior en su respectivo botón.

Una vez nos situamos en la página, o bien arrastramos desde el disco duro del sistema, o seleccionamos el área de *Drag and drop* y se elige el archivo CSV que se quiere añadir.

Una vez el archivo se ha seleccionado como se puede ver en la Figura A2.1, se pulsa el botón de "Send" y se espera hasta recibir un mensaje de confirmación de que se ha añadido con éxito.

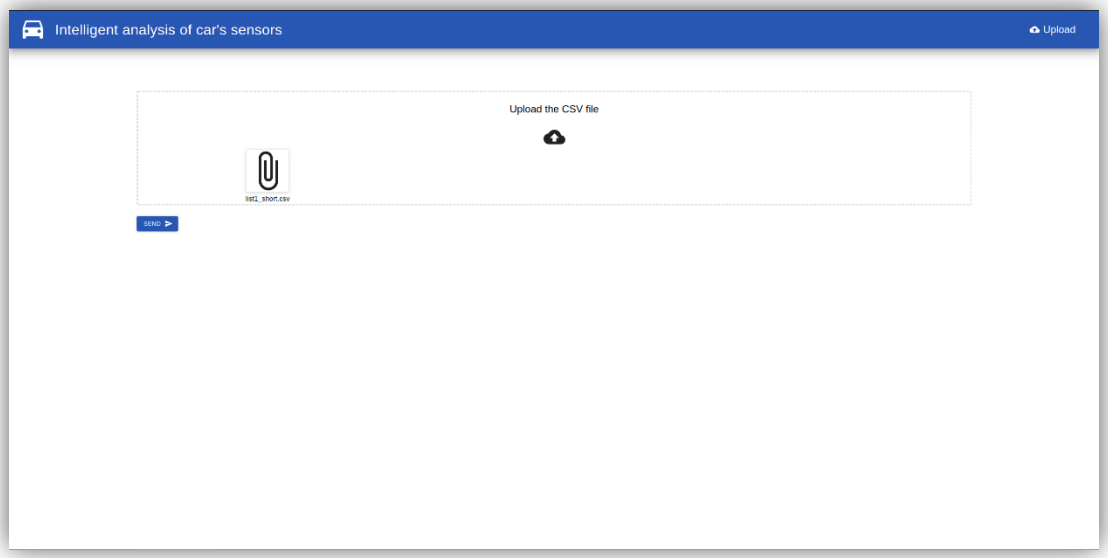


Figura A2.1 Página de "Upload".

Visualizar todos los datasets

Una vez se han cargado *datasets* en el sistema, para poder visualizarlos todos nos situamos en la pantalla de inicio. En ella encontraremos una tabla como aparece en la Figura A2.2 con todos los *datasets* añadidos y su información principal.

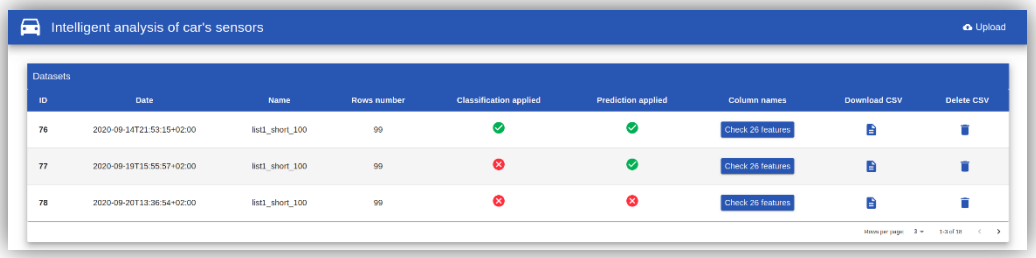


Figura A2.2 Pantalla principal mostrando tabla con todos los datasets del sistema

Eliminar *dataset*

Para eliminar un *dataset* determinado, nos situamos en la tabla de la Figura A2.2 y hacemos *click* en el botón de la columna "Delete CSV" como se puede observar en la Figura A2.3.

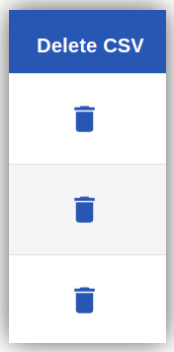


Figura A2.3 Icono para eliminar un dataset.

Descargar *dataset* en formato CSV

Para descargar el *dataset* en formato CSV nos situamos en la tabla de la Figura A2.2 hacemos *click* en el botón de la columna "Download CSV" como se puede observar en la Figura A2.4.

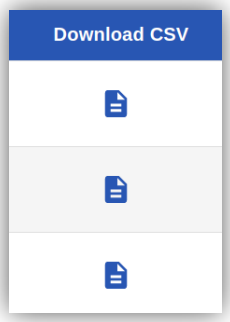


Figura A2.4 Icono para descarga dataset en formato CSV.

Consultar sensores para un *dataset* determinado

Para consultar los sensores o características de un *dataset*, que se corresponden con el nombre de las columnas que lo conforman, se hace click en el botón "Check 26 features" como se puede observar en la Figura A2.2 y se mostrará una ventana emergente como en la Figura A2.5 con un listado de cada característica acompañada de su unidad de medida. A su vez, si el ratón navega por encima de cada elemento se mostrará un mensaje de descripción.

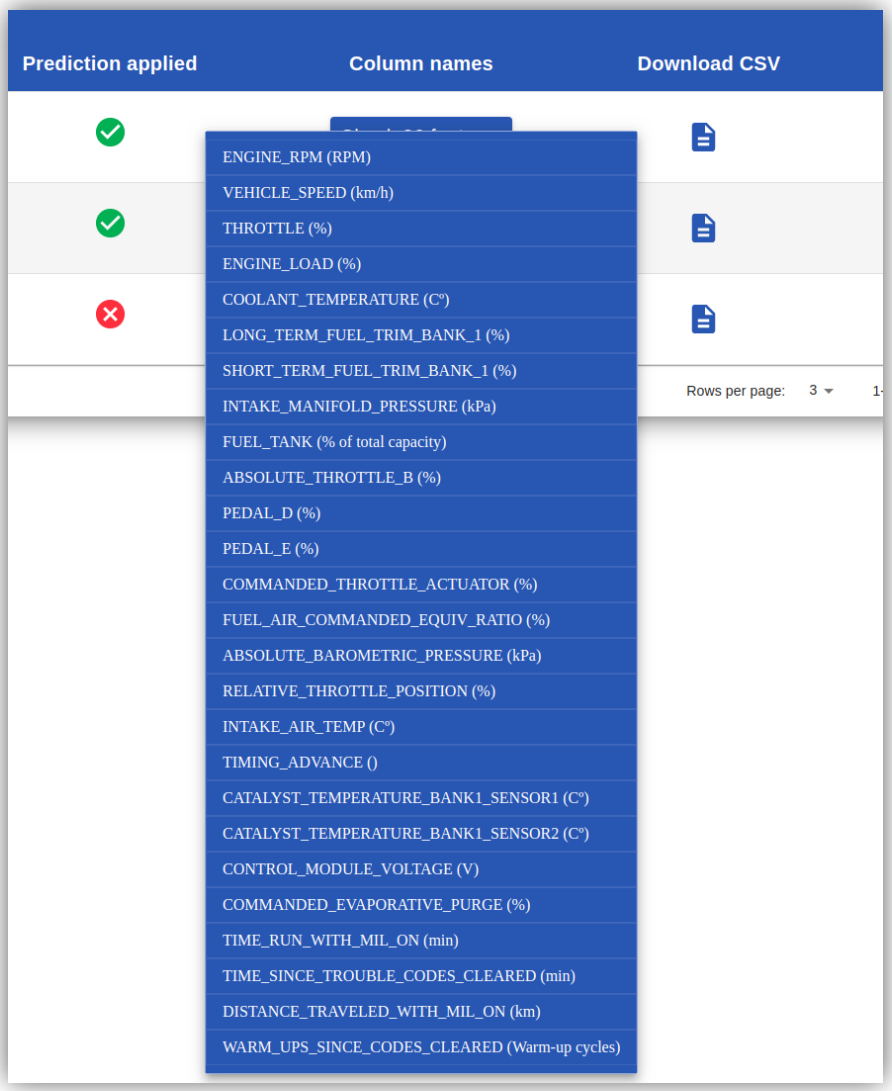


Figura A2.5 Listado con las características de un dataset.

Visualizar datos sobre cada sensor de un *dataset*

Una vez se ha consultado información más genérica sobre cada *dataset* usando la tabla de la Figura A2.2, es hora de consultar los detalles más específicos de cada uno. Si se desea visualizar la información de cada sensor a lo largo del recorrido del *dataset*, se tiene que hacer *click* sobre una fila de la tabla de la Figura A2.2 (correspondiente al *dataset* que queremos visualizar) y aparecerá en la parte inferior de la tabla un panel de detalles. En dicho panel, hacemos *click* en la pestaña "Dataset charts" y mostrará una sección con un diagrama de barras representando los datos. Como se puede observar

en la Figura A2.6, tiene un menú lateral izquierdo con un listado de todos los sensores existentes y si se hace *click* en cada uno muestra sus respectivos datos.

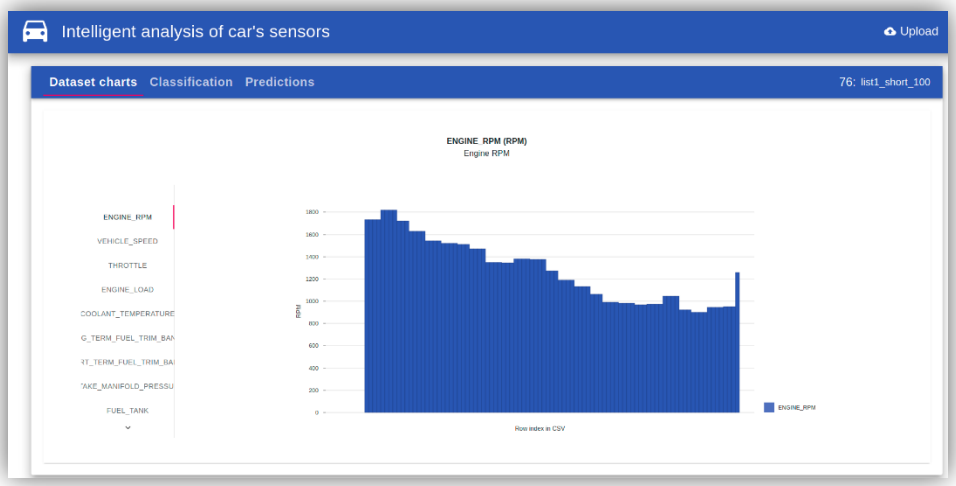


Figura A2.6 Diagrama de barras con los datos de cada sensor.

Aplicar clasificación sobre un *dataset* determinado

Para realizar una clasificación por medio de *k-means* y SVM, una vez se han cargado los detalles del *dataset*, nos situamos en la pestaña "Classification" (pestaña por defecto).

Antes de comenzar la predicción, como se puede observar en la Figura A2.7, se debe configurar dos parámetros: *Componentes number*, que se correponde con el número de componentes principales que queremos crear para PCA y *Clusters number*, que se corresponde con el número de categorías que se deseean formar. Una vez se han configurado, se hace *click* en el botón "Apply classification to dataset". Se mostrará un elemento de cargado mientras se espera la respuesta y un mensaje de éxito cuando se haya completado.

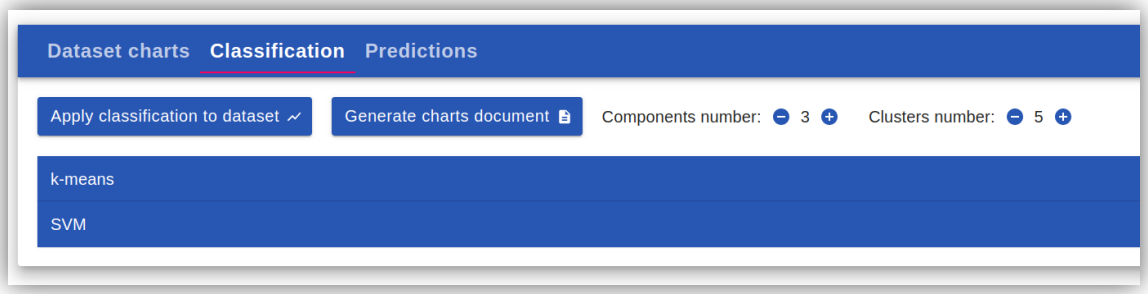


Figura A2.7 Panel de clasificación.

Generar PDF con los resultados de la clasificación

Una vez se ha realizado la clasificación, si se quieren obtener un PDF con las gráficas generadas, se tiene que hacer *click* en el botón "Generate charts document" de la Figura A2.7 y comenzará una descarga en el navegador.

Consultar información de la clasificación

Una vez se ha realizado la clasificación, si se quiere consultar los resultados de la clasificación tanto para *k-means* como para SVM, pulsamos en la sección deseada (Figura A2.7) y se abrirá una pequeña sección donde consultar la información detallada (Figura A2.8).

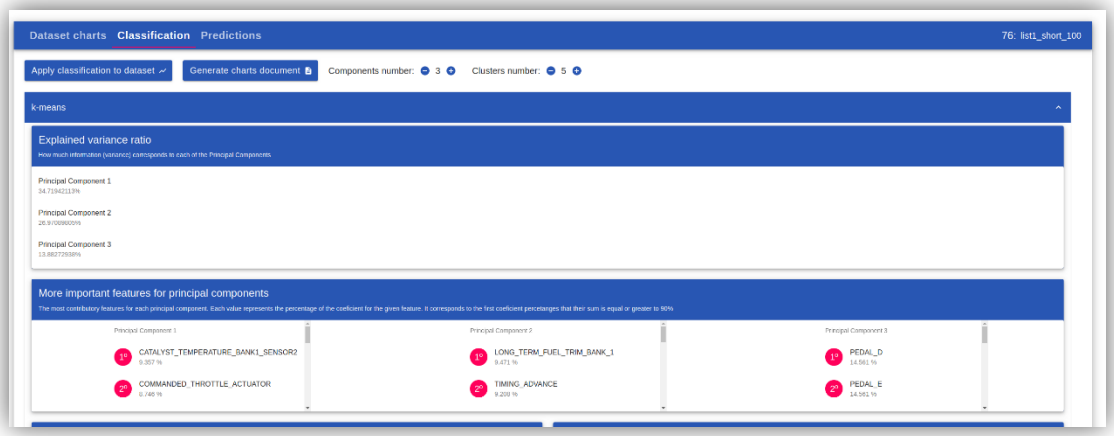


Figura A2.8 Resultados de *k-means*.

Aplicar predicción sobre un *dataset* determinado

Para realizar una predicción sobre un *dataset* concreto hay que haber seleccionado un *dataset* de la tabla (Figura A2.2) y que se muestre su información detallada. Se tiene que seleccionar la pestaña "Predictions" y se mostrará un panel como en la Figura 9. En él se puede seleccionar si se pretenden usar todas, una o aplicar PCA sobre las características, así como seleccionar la característica que se desea predecir y las épocas de la red neuronal. Una vez se ha configurado, se hace *click* en el botón "Apply prediction" y se mostrará un mensaje de cargado hasta recibir el resultado, el cuál se puede consultar en la parte inferior del formulario (Figura A2.9).

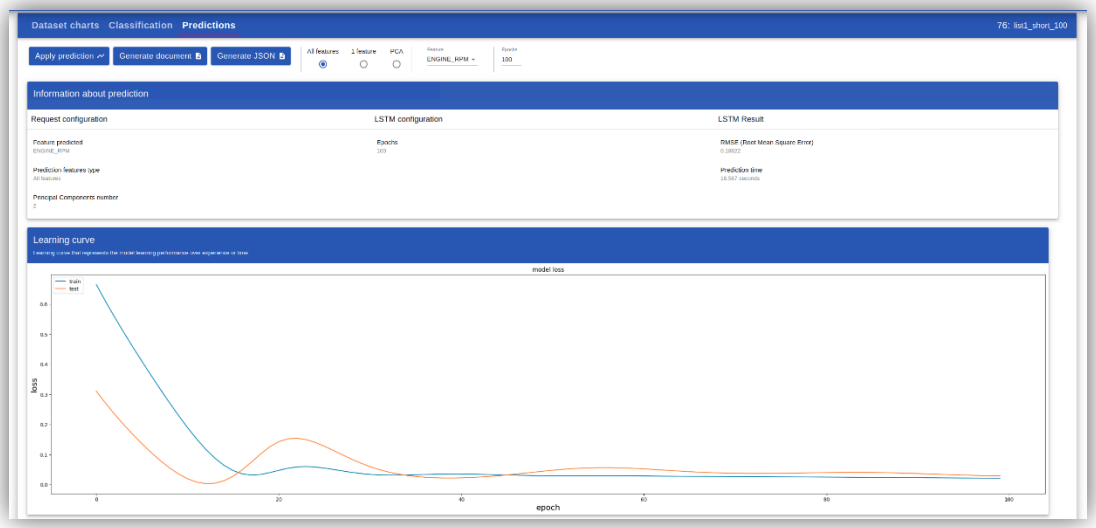


Figura A2.9 Panel de predicción.

Generar PDF con los resultados de la predicción

Una vez se ha realizado la predicción, si se quieren obtener un PDF con las gráficas generadas, datos de configuracion y resultados, se tiene que hacer *click* en el botón "Generate document" de la Figura A2.9 y comenzará una descarga en el navegador.

Generar JSON con los resultados de la predicción

Paralelamente a crear un PDF con los resultados de la predicción, también es posible generar un archivo JSON con la configuración e información de la predicción. Para ello se hace *click* en el botón "Generate JSON" de la Figura A2.9 y comenzará una descarga en el navegador.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA